AN INTERFACE TO THE ARPA NETWORK FOR THE CP/CMS TIME-SHARING SYSTEM. VOLUME 1

Joel M. Winett, et al

Massachusetts Institute of Technology

Prepared for:

Electronic Systems Division
Advanced Research Projects Agency

28 November 1973

AD773831

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Lincoln Laboratory, M.I.T.

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
An Interface to the ARPA Network for the CP/CMS Time-Sharing System - Volume I

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Technical Note

5. AUTHOR(S) (Last name, first name, initial)
Winett, Joel M. and Sammes, Anthony J.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
28 November 1973	116	20

8a. CONTRACT OR GRANT NO. F19628-73-C-0002
b. PROJECT NO. ARPA Order 600
c.
d.

9a. ORIGINATOR'S REPORT NUMBER(S)
Technical Note 1973-50 - Vol. I

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
ESD-TR-73-336

10. AVAILABILITY/LIMITATION NOTICES
Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES
Supplement to ESD-TR-73-337

12. SPONSORING MILITARY ACTIVITY
Advanced Research Projects Agency, Department of Defense

13. ABSTRACT

The interface to the ARPA network consists of a Network Control Program (NCP) which handles Host to Host communications, a LOGGER/SERVER for providing access to a CP virtual machine from the network, and a user subroutine package for communicating with other Hosts on the network from a logged on CP virtual machine operating in the CMS environment. The NCP and the LOGGER each run in separate virtual machines; the NCP handling the I/O operations with the IMP and the LOGGER handling pseudo I/O operations with CP through a software supported virtual terminal device. CMS virtual machines communicate with the NCP virtual machine through a special virtual machine to virtual machine communications facility.

This report describes the routines which make up the NCP and the LOGGER. Volume II includes flow charts on the logic for each of these routines.

14. KEY WORDS

ARPA network
time-sharing system
IBM 360/67
virtual machines

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

AN INTERFACE TO THE ARPA NETWORK FOR THE CP/CMS TIME-SHARING SYSTEM VOLUME I

J. M. WINETT
A. J. SAMMES
Group 23

TECHNICAL NOTE 1973-50

28 NOVEMBER 1973

LEXINGTON MASSACHUSETTS

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Advanced Research Projects Agency of the Department of Defense under Air Force Contract F19628-70-C-0230 (ARPA Order 600).

Table of Contents

Volume I

Appendices

Volume II: Flowcharts

List of Figures

List of Figures

(Continued)

An Interface to the ARPA Network

for the CP/CMS Time-Sharing System

1. Introduction

The interface to the ARPA network consists of a Network Control Program (NCP) which handles Host to Host communications, a LOGGER/SERVER for providing access to a CP virtual machine from the network, and a user subroutine package for communicating with other Hosts on the network from a logged on CP virtual machine operating in the CMS environment. The NCP and the LOGGER each run in separate virtual machines; the NCP handling the I/O operations with the IMP and the LOGGER handling pseudo I/O operations with CP through a software supported virtual terminal device. CMS virtual machines communicate with the NCP virtual machine through a special virtual machine to virtual machine communications facility.

All routines were written in assembly language for the IBM 360/67 system and operate with CMS in a CP virtual machine environment.

A brief description of the ARPA network is given, followed by an outline of the CP/CMS time-sharing system. This provides the fram work for a more detailed consideration of the interface software and the automatic initialization and error recovery features that are incorporated in the system design.

2. The ARPA Network

2.1 General Description

The ARPA Network is a store and forward, message processing network which interconnects a number of large, multi-processing computer systems throughout the United States. The computer systems, known as HOSTs, are linked together for the purpose of sharing resources among member sites. (See ROBERTS & WESSLER, Ref. 17). At each node of the network is an Interface Message Processor or IMP which controls all traffic through the node. (See HEART, KAHN, ORNSTEIN, CROWTHER & WALDEN, Ref. 9.) IMPs are interconnected by wide-band (50 kilobits per second) leased communication lines and provide for the connection to the network of up to four HOSTs at each site. The IMP is a slightly modified Honeywell DDP-516 computer and has been developed by BOLT, BERANEK & NEWMAN (Ref. 1).

The topology of the network, at the time the system was implemented, was as shown in Fig. 1 and Fig. 2. The computer systems connected were as listed in the table of Fig. 3.

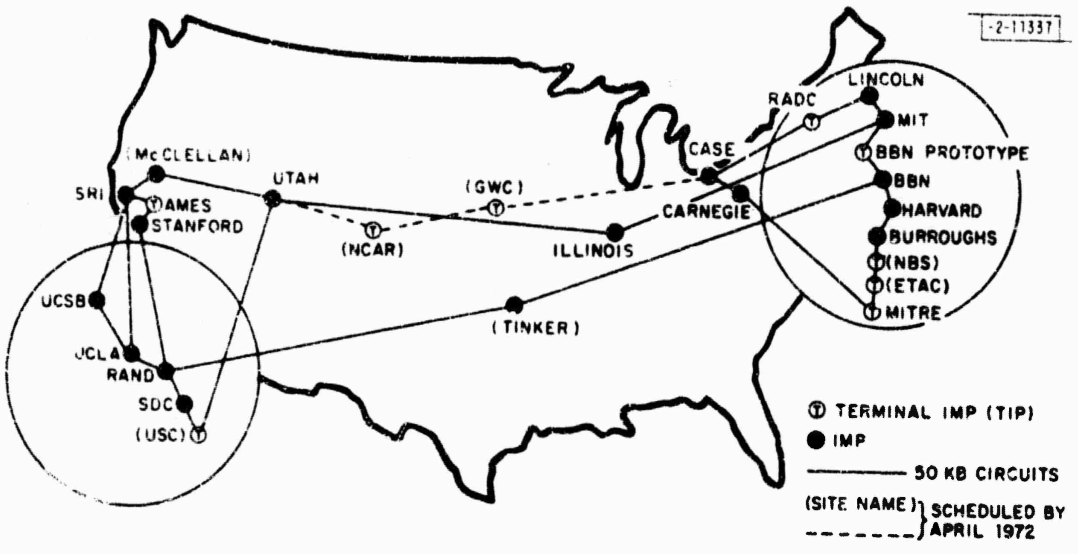


Fig 1: Network Topology - Fall 1971

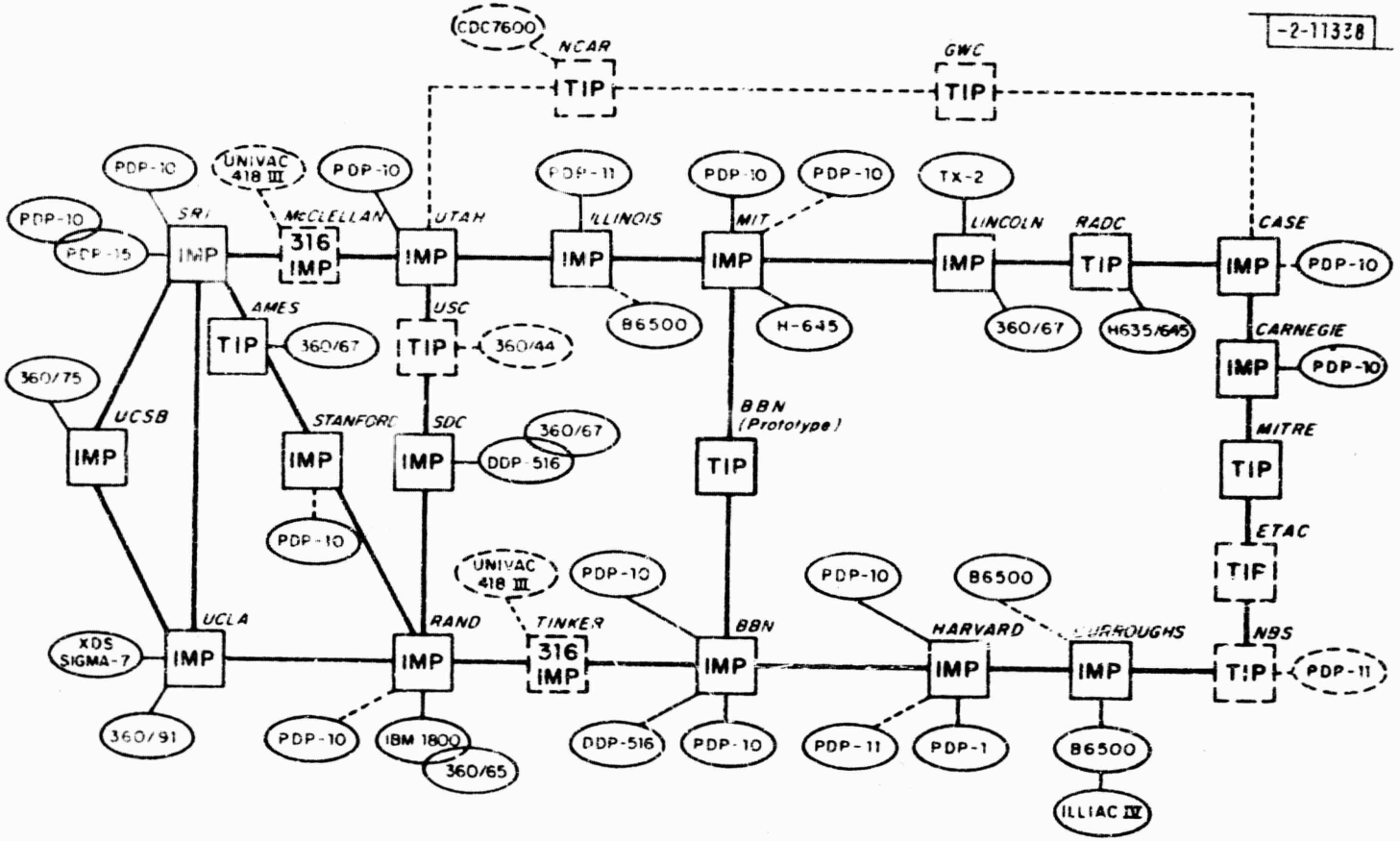


Fig. 2: ARPA Net Logical Map - Fall 1971

TN73-50-I(3)

SITE	HOST COMPUTERS	NETWORK IDENTIFIER
UCLA	Sigma 7 IBM 360/91	01 41
SRI	PDP 10	02
UCSB	IBM 360/75	03
UTAH	PDP 10	04
BBN	DDP 516 PDP 10 PDP 10	05 45 85
MIT	GE 645 PDP 10	06 46
RAND	IBM 1800	07
SDC	IBM 360/67	08
HARVARD	PDP 10	09
LINCOLN	IBM 360/67 TX-2 TSP	0A 4A 8A
STANFORD	PDP 10	0B
ILLINOIS	PDP 11	0C
CASE	PDP 10	0D
CARNEGIE	PDP 10	0E
PAOLI	Burroughs 6500	0F

Fig. 3: Hosts on the ARPA Network - July 1971

Since that time, the network has been expanded, and a more recent status report is given in KIRSTEIN (Ref. 10).

2.2 Network Concepts

The HOST computers typically differ from one another in type, speed, word length and operating system, though the notion of multiprogramming dominates most of the systems. Such HOSTs could concurrently support several users, with each user running one or more processes. A 'process', in this context, is taken to mean any software entity in a HOST with which an instruction counter is associated, and includes each concurrent activation of a re-entrant program. It corresponds closely to the term 'activity' used by SPOONER (Ref. 18).

A fundamental requirement of the design is to provide for process-to-process communication over the network. It was decided that such communication should be based, not on a solitary message, but rather upon a sequence of messages and that this sequence should conform to some standard formatting procedure or 'protocol'. A number of such process-to-process protocols have been defined and included:

a. An Initial Connection Protocol (ICP) which enables any process to gain access to specific processes (such as the system logger) at another HOST.

b. A Telecommunication Network (TELNET) protocol which enables any terminal serving process to be connected to any remote keyboard-printer terminal by mapping through a Network Virtual Terminal (NVT).

c. A Data Transfer protocol which specifies standard methods of formatting data for transmission through the network.

d. A File Transfer protocol which specifies methods for reading, writing and updating files sorted at a remote HOST.

e. A Graphics protocol which facilitates the transfer of graphics display information.

Processes which desire to communicate with one another have to be coupled by one or more 'connections'. A connection is defined to be simplex (uni-directional) and connects an output port of one process to an input port of another. Once such a connection has been established, messages passed to the output port of one process are automatically received at the input port of the other.

Difficulties exist in uniquely identifying ports (network-wide) in that each HOST, typically, has a different internal scheme for naming its processes. For this reason, a standard, intermediate name space is used, with a separate portion of the name space allocated to each HOST. The elements of the name space are called 'sockets' and each HOST is responsible for mapping sockets to ports for its own internal processes. A socket is specified by a 40-bit number, the first 8 bits of which identified the HOST. In the Lincoln Laboratory implementation, a further 24 bits are used to identify the internal process and the reamining 8 bits, the port within the process. The 'gender' of the socket (that is, whether it is send or receive) is marked by the last bit, thus giving each process a maximum of 128 send and 128 receive sockets.

Although it would be possible for processes to establish connections on their own behalf, it is considered desirable that control be co-ordinated in a standard manner, throughout the network, by a special purpose Network Control Program (NCP) resident in each HOST. Processes within a HOST then communicate with the rest of the network through their local NCP, which establishes, breaks, and controls data flow over connections as required.

In order to accomplish its tasks, the NCP of one HOST is required to communicate, in a well-defined manner, with the NCPs of other HOSTs. To this end a HOST-to-HOST protocol was defined by the Network Working Group (NWG), which specifies methods of establishing and breaking communications paths, the managing of buffer space, and a method of providing interrupt facilities. This protocol is known as the 'second-level protocol' and is transparent to the higher level process-to-process protocols outlined previously.

The medium for communication between HOSTs is the 'link'. Every HOST is considered to have 256 receive links from every other HOST, which the NCP could assign to connections as required. Links are permanently available to the NCP and are specified by the 8-bit HOST identifiers concatenated with an 8-bit link number. Link number 0 is used throughout the network as the 'control link', over which NCPs can issue 'control messages' to one another in accordance with the standards specified by the second level protocol. Control messages are always interpreted by an NCP as a sequence of one or more 'control commands' to effect such activities as the initiation of a connections, the assignment of buffer space, or the notification of an interrupt.

Each NCP maintains a 'rendezvous' (RV) table which records the current assignment of sockets to internal processes and links to connections. Incoming messages on a link, other than link 0, are passed by the NCP to that port of that internal process specified by an

appropriate entry in the RV table. If no such entry exists, the message is rejected. Outgoing messages, from an internal process, are associated with the appropriate link number, obtained from the RV table, before being transmitted to the appropriate HOST.

In order to effect any transmission of messages, via the network, an NCP has to communicate with its local IMP. A well defined procedure, known as the HOST-to-IMP or 'first level' protocol, has been specified for this purpose. Two types of messages are permitted by this protocol:

a. 'Regular' messages which are always associated with a link and which are passed by the IMPs, through the network, to a destination HOST. Such messages provide the mechanism for all HOST to HOST communication.

b. 'Irregular' messages which are communications between a HOST and its local IMP. Such messages are not, in general, passed through the network to a distant HOST, but are used for local control purposes.

The protocol is implemented through the medium of a 32-bit 'leader', appended to all messages. Fields in the leader specify the message type, the destination or source HOST and the link number. Regular messages are of message type 0, and irregular messages are of type non-zero, the particular value of the type defining the function of the irregular message. The protocol is transparent to the higher level HOST-to-HOST and process-to-process protocols referred to above.

Any regular message, passed by a HOST to its local IMP, is split, by that IMP, into one or more 'packets' for transmission through the network. Each packet is preceded by a 64-bit routing 'header' which contains such information as the destination and source HOSTs, the link number, the message and packet numbers and a set of flags.

A packet is passed, from IMP to IMP, through the network until it reaches its destination, as specified in its header. Each intermediate IMP selects the next leg of the transmission path, for the packet, on the basis of network topology, local traffic density and current line qualities. The destination IMP reconstructs the message from the individual packets, re-ordering them as necessary in accordance with the packet numbers. Such re-ordering is required because of the potentially different routes, and hence different transmission times, taken by each packet. When the messages have been reconstructed, the destination IMP passes it to the destination HOST and transmits an acknowledgement message to the source IMP. This acknowledgement message is known as a Ready For Next Message (RFNM). On

receipt of the RFNM, the source IMP advises the source HOST that the message had been passed and that the link associated with it is now free for further transmissions. The packet mechanism and the IMP to IMP communication procedures are transparent to all the higher level protocols.

At the hardware level, a HOST is connected to its IMP by a specially designed interface, across which information can be transmitted, bit serially, in either direction. It is this interface which resolved the differences in word lengths and speeds between the HOSTs and IMPs. Transmissions consist of a bit train, containing no special indication of word or byte boundaries, but interrupted as required while the sender fetches a new word from memory, or the receiver writes an assembled one into storage.

The Lincoln Laboratory 360/67 is connected to the IMP through a specially built interface box. This box is attached as a control unit on the multiplexer channel and provides a duplex connection to the IMP. Thus the IMP appears as two devices, one for transmission of data and one for the reception of data, both of which can operate simultaneously. The specifications of data transfer to and from the IMP through this interface box is described in Appendix A.

2.3 Requirements of the HOST Software

The network control software, resident in each HOST, is required to observe five major interfaces and effect the protocols associated with them. These are:

a. The hardware interface between a HOST and its IMP, and the protocol which specifies the control of that interface.

b. The software interface between a Network Control Program and its local IMP, and the HOST-to-IMP protocol.

c. The software interface between Network Control Programs, and the HOST-to-HOST protocol.

d. The software interface between and internal process and its local Network Control Program, and the protocol which describes the form of messages between them.

e. The software interface between processes, and the various process-to-process protocols.

Of these protocols, the HOST-to-IMP, HOST-to-HOST and process-to-process are network defined; the remainder are implementation dependent and vary in detail from site to site. An outline description of the network

defined protocols follows; the implementation dependent ones are described in Section 4.

2.4 HOST-to-IMP Protocol

The HOST-to-IMP protocol was specified by BOLT, BERANEK & NEWMAN (Ref. 2) who designed the IMP software and modified the Honeywell DDP-516 computer. This protocol is based on the concepts of a 'message' and a 'link'.

A message consists of not more than 8096 bits of data which includes 32 bits for a leader and at least one bit for 'padding'.

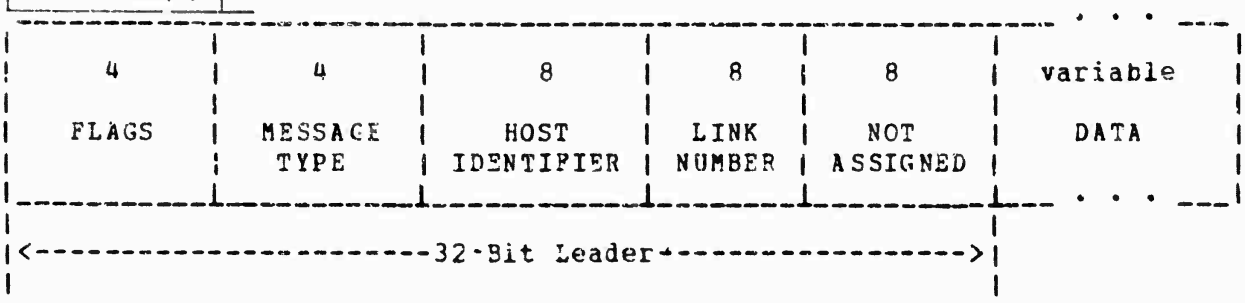
Padding is added by the hardware interfaces to resolve the word length mis-match problem of dissimilar HOSTs and IMPs. The source HOST to IMP interface appends a single one bit to the end of a message, followed by sufficient trailing zero bits to complete an integral number of IMP words. The destination IMP to HOST interface appends additional trailing zero bits to complete an integral number of destination HOST words. The end of the message can thus be located by a destination HOST by searching its input buffer, back from the last word transferred, until it finds a non-zero bit. The bit prior to this will be the last bit of the message sent by the source HOST. No network significance is to be placed on HOST word or byte boundaries.

Two classes of message type are specified, as stated previously; regular messages which have a zero message type and which provided the medium for all HOST to HOST communications; and irregular messages which have a non-zero message type and which are used for control purposes between a HOST and its IMP. (It should be noted that messages of non-zero message type are referred to in BOLT, BERANEK & NEWMAN (Ref. 2) as 'control messages'. This term is also used in the HOST-to-HOST protocol for regular messages on link zero. To avoid ambiguity, the term 'irregular message' is used throughout this report to refer to messages of non-zero message type, and the term 'control message' is reserved for HOST-to-HOST protocol use.)

The 32-bit leader determines the type of the message and the link, if any, with which it is to be associated (Fig. 4).

Regular messages are always to be associated with a link and this is fully specified by a destination HOST identifier, a source HOST identifier, and a link number. Of these, the originating HOST specifies, explicitly, only the destination HOST identifier in the HOST identifier field, and the link number. At its destination, the destination IMP regenerates the leader from the packet header, replacing the value originally in the HOST identifier field by the source HOST

TN73-50-I(4)



HOST TO IMP MESSAGES		IMP TO HOST MESSAGES	
MESSAGE TYPE	MEANING	MESSAGE TYPE	MEANING
0	Regular message HOST to HOST.	0	Regular message HOST to HOST.
1	Error in leader.	1	Error in leader.
2	HOST going down.	2	IMP going down.
3	-	3	Blocked link.
4	No operation.	4	No operation.
5	Regular message for discard.	5	RFNM.
6	-	6	Link table full.
7	-	7	Destination IMP or HOST dead.
8	Error not in leader.	8	Error not in leader.
9	-	9	Incomplete transmission.
10-15	-	10-15	-

Fig. 4: HOST-to-IMP Message Format

identifier. Once a message had been assigned to a link, the link is said to be 'blocked' and the IMP will refuse further transmissions over it until the first message has either been delivered or rejected. Notification of these events is by irregular IMP to HOST messages; a type five indicating successful delivery (RFNM); a type seven, that the destination HOST did not receive the complete transmission. Each of these messages unblocks the link. Other irregular messages in the protocol enable an IMP to indicate to its HOST that a link is blocked; that errors have been noted in previous messages; or that its link table is full. The HOST can indicate to its IMP that it is closing down; that errors have been noted in previous messages; or that a particular message is for test purposes only. The messages are summarized in Fig. 4 and described in more detail in Appendix B and C.

With reference to the leader, the FLAGS field, of four bits width, is set to zero by source HOSTs and ignored by destination HOSTs. The field is to be used only to communicate with IMP background programs for diagnostic, maintenance or measurement purposes. The fields, MESSAGE TYPE (four bits), HOST IDENTIFIER (eight bits) and LINK NUMBER (eight bits) are to be used as described above. The remaining eight bits of the leader are unassigned. The DATA field, of variable width, is unused in irregular messages and is formatted according to the HOST-to-HOST protocol conventions in regular messages.

The IMPs exercise traffic control and restrict the total amount of data in transmission through the network by:

a. Refusing additional messages over a 'blocked' link.

b. Limiting the maximum size of a message to 8096 bits.

c. Limiting to 63 the number of transmit and receive links a HOST can concurrently have active.

d. Rejecting any message which required longer than 15 seconds for re-assembly from its constituent packets.

e. Rejecting any message which has not been collected by a destination HOST within 40 seconds of re-assembly.

The constraint of a regular message to not more than 8096 bits in length is imposed by the IMP sub-network. The second-level or HOST-to-HOST protocol passes this constraint to the user process.

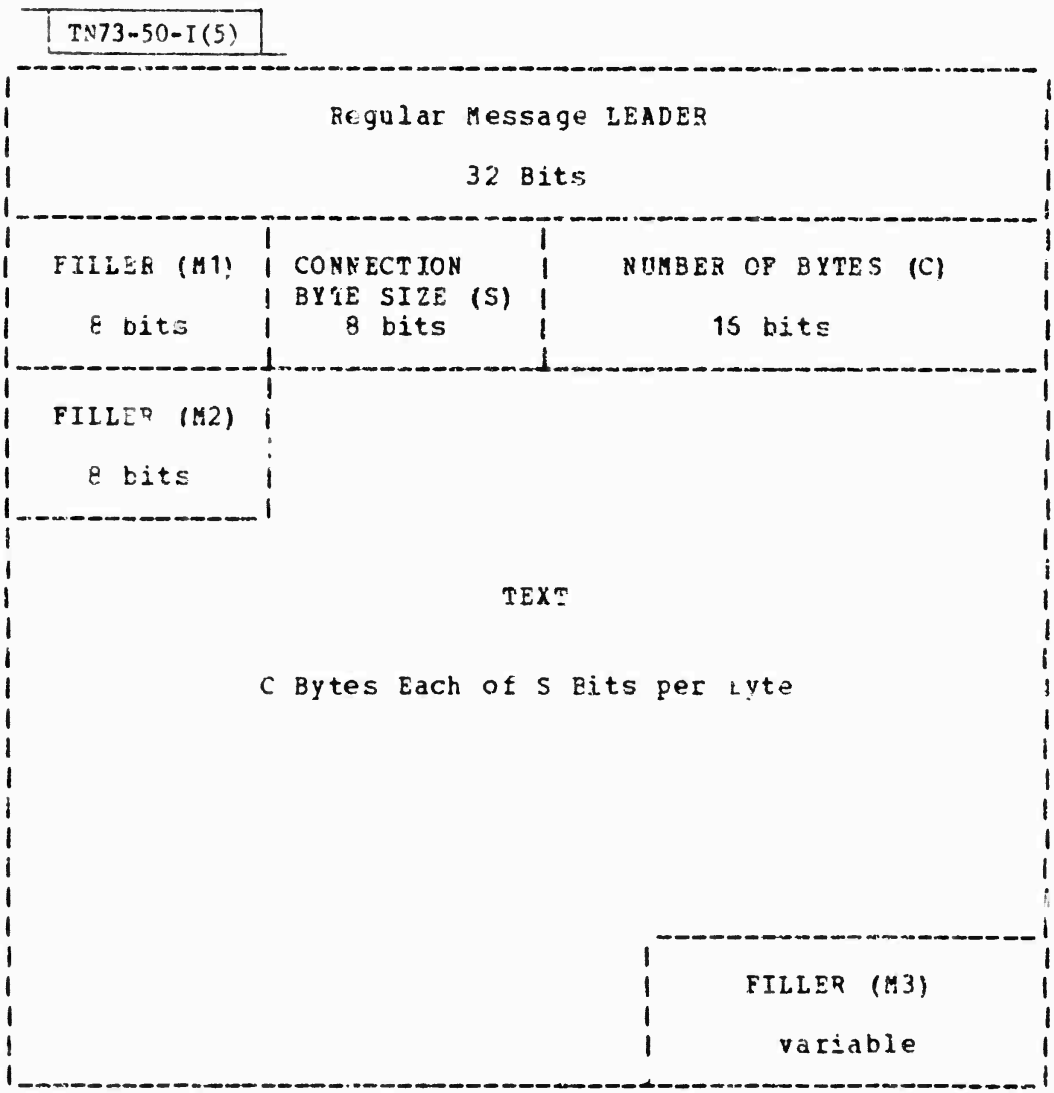


Fig. 5: HOST-to-HOST Message Format

2.5 HOST-to-HOST Protocol and the NCP

An early design for the HOST-to-HOST protocol was reported on in CARR, CROCKER & CERF (Ref. 4), but the version implemented was formulated by the Network Working Group and is specified in CROCKER (Ref. 7).

This protocol is based on the concept of a HOST-resident Network Control Program (NCP), which can communicate with other HOST-resident Netsork Control Programs, by means of 'control messages', issued over 'control links', for the purpose of effecting 'connections' between 'sockets' assigned to HOST-resident processes.

All HOST-to-HOST communications are regular messages, control messages being sent over link zero and interprocess transmissions over non-zero links. The transmitting NCP is responsible for segmenting interprocess transmissions into regular messages, not more than 8096 bits in length. The receiving NCP is therefore responsible for concatenating successive regular messages into a single, or differently divided, transmission for delivery to the receiving process. Because of the differing word lengths of the communicating HOSTs, the DATA field of regular messages (Fig. 4) will not, necessarily, start or end on word boundaries at the receiving HOST. In order to concatenate successive regular messages, the receiving NCP has to be prepared to employ extensive bit shifting; an expensive operation in terms of processor time. To alleviate this situation, the notions of a 'message header' and a 'connection byte size' have been introduced.

The message header embraces the regular message leader and extends the required information at the beginning of each message to a total of 72 bits. The figure of 72 was chosen as the lowest common multiple of most HOST byte and word lengths; these being 8, 18, 24 and 36 bits. This header reduces the bit shifting problems, at the beginning of a message, for the majority of HOSTs. The format of the header is given in Fig. 5.

It consists of the 32-bit leader, specified by the HOST-to-IMP protocol; an 8-bit filler field (M1); an 8-bit connection byte size field (S); a 16-bit number of bytes field (C) and a further 8-bit filler field (M2). The text field of a message consists of C bytes each of S bits per byte. It was decided that, for each connection, a connection byte size should be agreed upon, by the communicating procedures, and that this would typically be the lowest common multiple of the respective HOST word lengths. The text of messages would then be an integral number of these bytes. This procedure reduces the bit shifting problems at the end of a message. In addition, a variable length filler field (M3), is specified at the end of a message which can be used by the transmitting HOST to fill the message

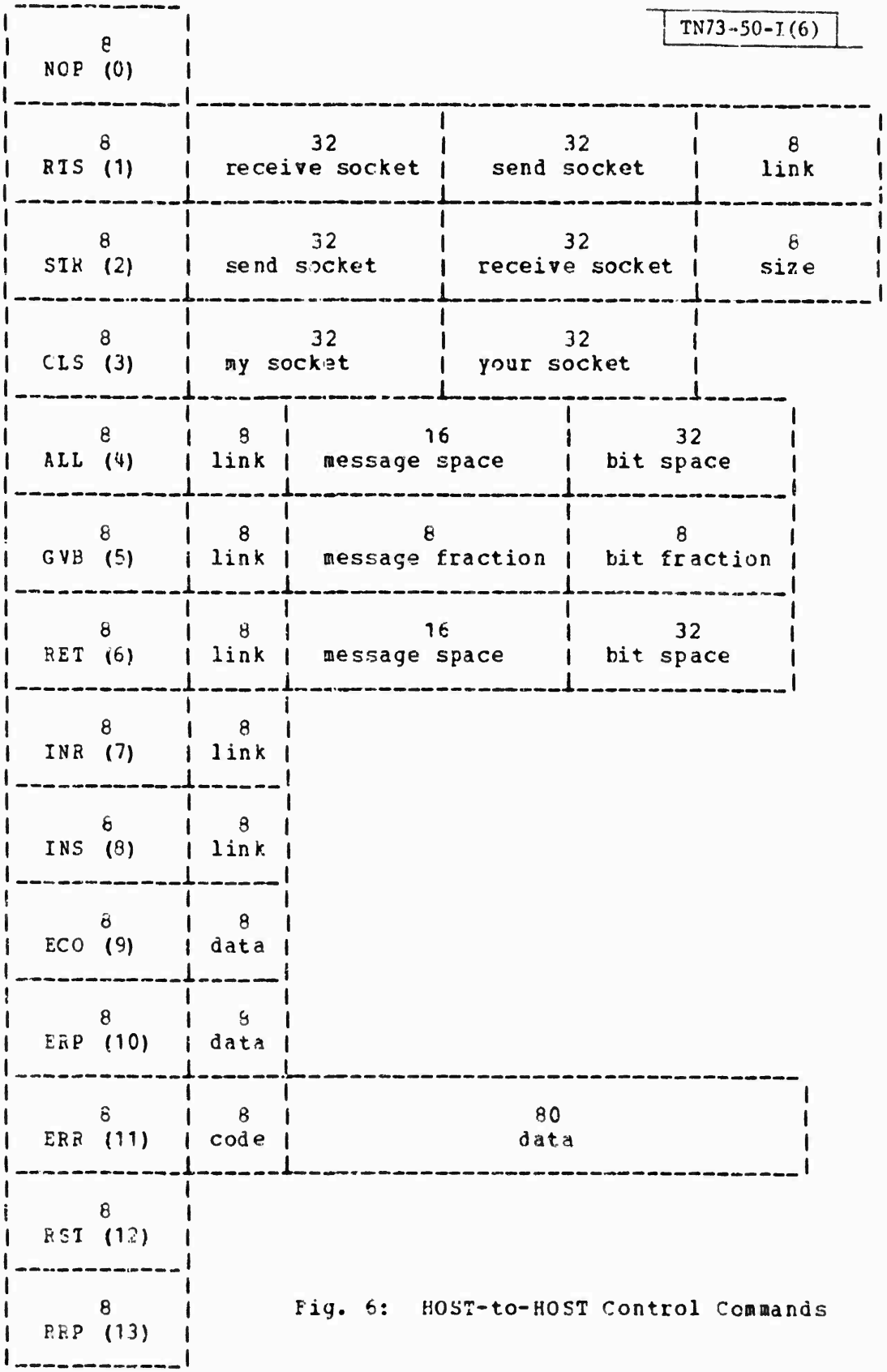


Fig. 6: HOST-to-HOST Control Commands

out to a word boundary. All the filler fields have a zero value.

A connection byte size is specified when the connection is first established, on behalf of processes, by the co-operating NCPs. Connections between NCPs , on links number zero, are not, however, explicitly established but are considered to exist when a HOST becomes active. The connection byte size for these connections is implicitly defined to be 8 bits, and the text field always consists of an integral number of control commands.

The format for each control command is given in Fig. 6; the first field of 8 bits, known as the 'op-code', defines the function of the command and successive fields specify parameters to it. The figure shows the three character mnemonic name of each op-code; its numeric value in parentheses; the size of the parameter fields and the type of parameters required. Op-codes are defined for such functions as the establishment of a connection, RTS (receiver to sender) and STR (sender to receiver); the termination of a connection, CLS (close); the control of buffer space, ALL (allocate), GVB (give back) and RET (return); the notification of interrupts, INR (interrupt receive link) and INS (interrupt send link); the reporting of errors, ERP (error) and the determination of status, ECO (echo), ERP (echo reply), RST (reset) and RRP (reset reply).

When two processes desire to establish a connection, the prospective sender requests its NCP to issue an STR command and the prospective receiver requests its NCP to issue an RTS command. The STR command specifies the local send socket, the foreign receive socket and a connection byte size. The RTS command specifies the local receive socket, the foreign send socket and a previously unassigned link number for use by the connection. Once these commands, known as 'requests for connections' (RFCs), have been exchanged, and provided that the specified sockets match and were not already in use, the NCP of the prospective receiver issues an ALL command specifying the amount of buffer space it is prepared to assign to the connection. On receipt of this command, the NCP of the prospective sender is permitted to transmit data, on behalf of its process, up to the amount specified in the command. It is expected that as the NCP of the prospective receiver passes the data to its process and releases its internal buffer space, it will issue further ALL commands to the sender, to enable transmissions to continue. The NCP of the prospective receiver can, at any time, request a return of all or part of the buffer allocation by means of a GVB command and this is to be acknowledged by a RET command. All NCPs are to maintain message and bit counters for each of their send connections which are to reflect the current buffer allocation at the receiving site. They are specifically prohibited from issuing any message

which would result in either counter being decremented below zero. This flow control mechanism does not apply, however, to control messages issued on links number zero.

Connections can terminate by either NCP issuing a CLS command which the other NCP is to echo. Care is required to ensure that a CLS command, issued by a sending NCP on link zero, does not arrive at the receiving NCP before the last data message on the connection to which it refers.

An NCP is permitted to issue an ERR command whenever it detects a HOST-to-HOST protocol error by another NCP. The code field of this command specifies the type of the error and the data field gives additional information about it. A number of error types are network defined but it is expected that implementers of NCPs would publish details of their own additional error types. (See Section 4).

Further details of the HOST-to-HOST protocol are given in Appendix D.

2.6 Process-to-Process Protocols

Two process-to-process protocols were defined at the time the system to be described was implemented; the Initial Connection Protocol (ICP) and the Telecommunication Network protocol (TELNET). These protocols are used both by the LOGGER SERVER and the user TELNET processes discussed in Section 4.

A family of ICPs is formally specified, on behalf of the Network Working Group, by POSTEL (Ref. 15). The protocols require a 'listen' function to be defined which permits a process to specify to its local NCP the identity of a socket for which it is prepared to accept a connection from any other process. On receipt of an appropriate request for connection to such a socket, the local NCP is to issue the matching RFC, thus completing the connection, and to advise its process accordingly. This function is effected in the Lincoln Laboratory system by the process-to-NCP command, enable connect (ENC) and enable listen (ENL) described in Section 4.

Using this protocol, a server process listens on one of its send sockets, L, which is well known throughout the network. A user process, wishing to communicate with the server process, requests its NCP to establish a connection to L from one of its receive sockets, U. On establishment of the connection, which is to have a byte size of 32 bits, the server process has two options. It could either request its NCP to close the connection, indicating that it is not prepared to communicate further with the user process, or, it could send a single 32-bit even number S, identifying a pair of sockets, S and S+1, which are to be used for all

subsequent transmissions. If the server process adopts the latter option, both processes are to request their respective NCPs to close the connection between L and U and to establish new connections between S and U+3 and S+1 and U+2. The protocol thus provides a means of establishing duplex communications between process. The well known socket, L, is called the Initial Connection Socket and is defined to be X'00000001' for the LOGGER SERVER, where X signifies a hexadecimal number. U+2 and U+3 are specified for use, rather than U and U+1, in order to avoid a race condition.

The TELNET protocol, described by O'SULLIVAN (Ref. 14) on behalf of the TELNET committee, was designed to make a process at a using site appear to a process at a serving site as logically equivalent to a terminal directly connected to the serving site. The protocol is effected through the medium of a Network Virtual Terminal (NVT) which normally uses a standard, intermediate terminal code. Both the serving and using processes map their local codes to this intermediate code and must conform to the conventions of the NVT. This approach eliminates the need for a HOST to keep information about the characteristics of terminals and terminal handling conventions at other host sites.

The intermediate data code of the NVT is defined to be the 7-bit ASCII code transmitted in 8-bit bytes with the high order bit set to zero. Connections between processes are established using the Initial Connection Protocol and have a connection byte size of 8 bits. A number of 8-bit TELNET control codes are also defined, each with the high order bit set to one. These are used for such purposes as the notification of a break or reverse break signal; a request to echo or cease echoing data characters, or a warning to suppress local printing of input characters. In addition, a data type control code can be sent as the first byte of data over a connection to signify whether the standard NVT code is to be used or some defined alternative such as EBCDIC. A special synchronising code is also defined, which can be inserted in the data stream and which is always associated with a network interrupt (INS) command. This enables stacked input to a serving process, up to the synchronising code, to be discarded or temporarily ignored by those systems which recognised an 'attention' key as having a specified interrupt function.

3. The CP/CMS Time-Sharing System

The Lincoln Laboratory CP/CMS time-sharing system is a modified version of IBM's CP-67/CMS time-sharing system (Ref. 6) and consists of two independent components: the Control Program (CP) and the Conversational Monitor System (CMS). The Control Program creates a set of 'virtual machines' which are time shared with one another, and the Conversational Monitor System provides

a conversational operating system which can be used to control a virtual machine.

A virtual machine is a functional simulation of the real IBM 360/67 and its associated input-output devices. The virtual memory and virtual devices of a virtual machine are specified by a predefined configuration which can differ for each virtual machine. The Control Program allocates the CPU resources of the real machine, for a short period of time, to virtual machines in turn, in accordance with a scheduling algorithm.

All input/output commands from a virtual machine are trapped by CP and translated to real I/O operations. Unit record input-output is normally spooled onto disk. Virtual memory is divided into 4096-byte blocks, held on paging drums and paged in and out of real memory as required. Special hardware is provided on the IBM 360/67 for paging purposes which performs 'dynamic address translation'.

CP maintains a record of all permitted virtual machines and identifies each one by means of a USERID. Associated with this USERID is a password, accounting information and a table describing the configuration of the related virtual machine. Provided that the number of active virtual machines is not the maximum that the system can support at any one time, a user could request CP to establish his virtual machine by logging in on an unassigned terminal. "Logging in" involves the typing of a USERID and password which are used to validate an authorized user. The terminal enables a user to control his virtual machine, by means of CP console functions, in a manner similar to that of an operator at the console of the real machine. The virtual machine appears to a user and his programs to be indistinguishable from the real machine.

Having established the virtual machine, the user can employ CP console functions for such purposes as the Initial Program Loading (IPL) of an operating system; the display or modification of any portion of his virtual memory; The stopping or restarting of his virtual machine; the sending of messages to other terminal users, the setting of special virtual machine facilities; and the querying of CP for systems information.

CMS is one of the operating systems which can be loaded into a virtual machine by means of the IPL function. It is a single user conversational system which provides a comprehensive command language and programming facility. Commands exist for file creation and manipulation, program compilation, execution control, debugging and various utility operations.

The file handling commands enable the user to create, copy, move, combine, edit, print and erase disk files

and other commands provided access to tape units, line printers, card readers and card punches. Compilation commands are available for Assembler, Fortran IV and PL/1, and the execution control commands allow the user to load relocatable object programs and execute them under terminal control. Executing programs can be halted at pre-determined points, examined, modified and restarted using the debug commands. Utility functions provide for tape and disk copying, sorting, dumping and printing.

An EXEC facility which includes substitutable parameters and conditional statements allows a set of commands, stored in a file, to be obeyed by the command interpreter. Control can be transferred from the CMS command interpreter to the CP console function environment by means of an 'attention' key, thus allowing looping programs to be aborted.

Both CP and CMS commands can be issued from the terminal or called from user programs. All the network software was designed to operate in a virtual machine environment, under control of the CMS operating system.

4. The Lincoln Laboratory IBM 360/67 Network Software

Support for the ARPA network on the IBM 360/67 computer at Lincoln Laboratory is provided through a Network Control Program (NCP) which runs in a virtual machine of the CP-67 operating system. The NCP virtual machine has direct access to the Interface Message Processor (IMP), and hence to the network, and communicates with other virtual machines over a pair of virtual devices, one for sending and one for receiving. The virtual device used for sending can be directed to a particular virtual machine in the same way that user virtual machines direct transmission to the NCP virtual machine. Transmissions to the NCP are identified as to the sending user and thus are appropriately processed by the NCP.

The implementation of the NCP in a virtual machine rather than as part of the core resident system facilitated its development since it could operate independently of other virtual machines and, hence, not impact the reliability of the operating system. The NCP is written as a user program which calls on CMS system facilities for program initialization, including program re-initialization in the event of a catastrophic error, for disk file services, and for virtual machine to virtual machine communications.

By operating the NCP in a virtual machine, core memory is not dedicated to support the network operation; system resources are only required when network activity occurs. In addition, operating the NCP in a virtual machine is particularly convenient since the protocol by

which NCP's communicate is being changed and will continue to evolve. The added overhead to the operation of the NCP caused by these techniques does not preclude useful network operation and experimentation. A number of improvements can be implemented to reduce this overhead by giving the NCP special priorities if the use of the network is limited by the performance of the NCP.

The NCP keeps a table (the rendezvous table - RV table) whose entries reflect the state of connections between the local NCP and a foreign NCP. Each entry contains the identification of a foreign socket and a local socket, and the userid of the local virtual machine which is to be associated with the local socket. Entries are made in the RV table either on the receipt of a Request For Connection (RFC) control message from a foreign NCP or through a request from a local virtual machine.

The communication protocol between a user virtual machine and the NCP resembles the NCP to NCP protocol. Data cannot be transmitted until a connection is initiated by both the sender and the receiver and the NCP to NCP requirements for flow control are met. The NCP specifies the number of bits it is prepared to buffer for a receive connection and stores messages received from a foreign NCP until a local user makes a request for a message. If no message has been received when a user makes a receive request, this fact is returned to the user. The NCP will not accept a message from a user for tranmission if the receiving NCP has not provided adequate buffering. In this way, the requirements for flow control between NCP and NCP are passed to the user. The NCP also notifies the user of any asynchronous change in the state of a connection so that he can take appropriate action. For example, the user is notified a) when a connection he has initiated has been completed by the foreign host, b) when a message is received and there had not been any previous messages buffered, and c) when a connection is closed by the foreign host. In addition, the user is notified when a network interrupt has been received for the connection.

A set of low-level user routines (NET) have been provided to communicate with the NCP. These routines follow the protocol defined for network communications through the NCP. For each user request to the NCP, a reply from the NCP is sent to the user virtual machine indicating either successful or unsuccessful completion of the request and the current status of the connection. These routines handle the stacking of asynchronous interrupts which are unstacked through a call to the NET routines.

One user virtual machine which has been developed is the LOGGER virtual machine. The LOGGER controls a number of virtual terminals which provide access to the CP system

in a manner similiar to physical terminals, but instead of a hardware terminal, CP is interfaced with a specially designed virtual terminal device which is driven by software. Thus, information normally typed on a terminal by CP and information normally keyed into a terminal is transferred between the LOGGER and CP through one of these virtual terminal devices. The protocol for controlling a software terminal is described in Appendix E.

When the LOGGER is initialized it is prepared to communicate with the NCP over the socket defined for LOGGER operation under the Initial Connection Protocol (ICP). Communication under the ICP will result in a new pair of sockets being defined through which the LOGGER will provide CP services to the network. Once this pair of sockets is defined, the LOGGER acts as a SERVER interfacing a virtual terminal with a remote process through the NCP.

During the initial login sequence, the LOGGER analyzes received network messages for 'userid' and 'password' and compares these with entries in a special file. If the userid does not appear in the file, an invalid userid is passed to CP. If the password matches the network password corresponding to the valid network userid specified in the file, the system password is sent to CP. Since network passwords and valid system passwords do not have to be identical, access to the CP system from the network is specially controlled and may be restricted to a subset of authorized system users. Though the operation of the LOGGER/SERVER as a separate virtual machine introduces another level of dispatching overhead, this does not appear to be significant and does not limit its use.

4.1 System Design

Figure 7 outlines the major elements of the network software and the interfaces between them.

The Network Control Program (NCP) resides in a virtual machine (VM1 in the figure) to which is attached the IMP hardware interface and a terminal. The terminal is used for establishing, and monitoring the activity of the NCP. Processes in other virtual machines communicate with the NCP through the CP virtual machine to virtual machine communications (VMCOM) interface. This facility was specially designed for efficient virtual machine to virtual machine communications.

In an earlier version of the network software, the CP facility for transferring card image from the virtual card punch of one virtual machine to the virtual card reader of another virtual machine was used. This process used the CP spooling facility which limited the rate of virtual machine inter-communications to the speed of writing a spooled file to the disk and then reading the

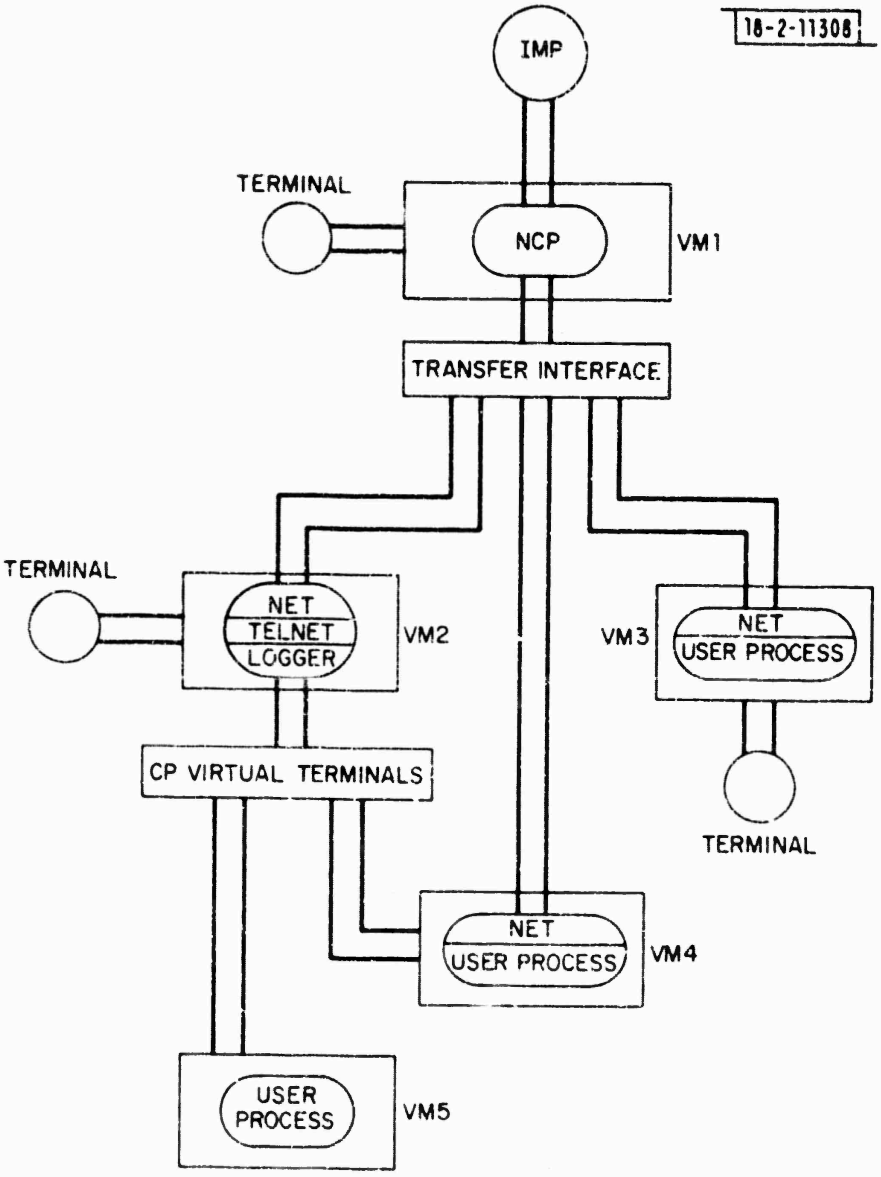


Fig. 7: System Design

spooled file. With the VMCOM facility, data is moved from the virtual memory of one virtual machine to a CP free storage area, and then from the CP free storage area to the virtual memory of another virtual machine. A description of the VMCOM facility is contained in Appendix F.

In Figure 7, three virtual machines are shown connected to the NCP through the VMCOM interface. VM3 represents a virtual machine, controlled by a user at his terminal, executing a process which is in communication with the network. All user processes communicate with the NCP by means of a subroutine and macro package, NET, which effects a locally defined process-to-NCP protocol.

VM2 represents a virtual machine containing the logger-server process. This process uses the NET package to communicate with the NCP and observes the TELNET and ICP process-to-process protocols for all network transmissions. A modification was made to CP, by the system maintenance group, to provide a CP virtual terminal interface for use by the logger-server. This interface enables the logger-server to establish and service virtual machines as though it, itself, was a set of local terminals each controlled by a separate user.

The figure shows two virtual machines, VM4 and VM5, established from the network, by the logger-server, through the virtual terminal interface. Both virtual machines contain user processes which are executing under the remote control of network users or their local processes. In addition, VM4 represents the case where the executing user process is itself accessing the network by means of the NET package.

This design allows for more complex configurations to be established than that shown in the figure. Any number of active virtual machines could use the NET/VMCOM interface and each user process is permitted a maximum of 256 connections to the network. The logger-server currently supports up to three virtual machines, at any one time, as this is considered to represent the maximum, in terms of local resources, that should be allocated to the network.

Brief descriptions of each of the major elements follow, together with some implementation details.

4.2 The NET Package

Figure 8 shows in block schematic form the way in which a user process communicates with the Network Control Program.

A set of high-level NET commands are defined which can be employed by a process to call subroutines which used the NET package. Each high-level command is in the form of a macro, which expands into one or more primitive NET

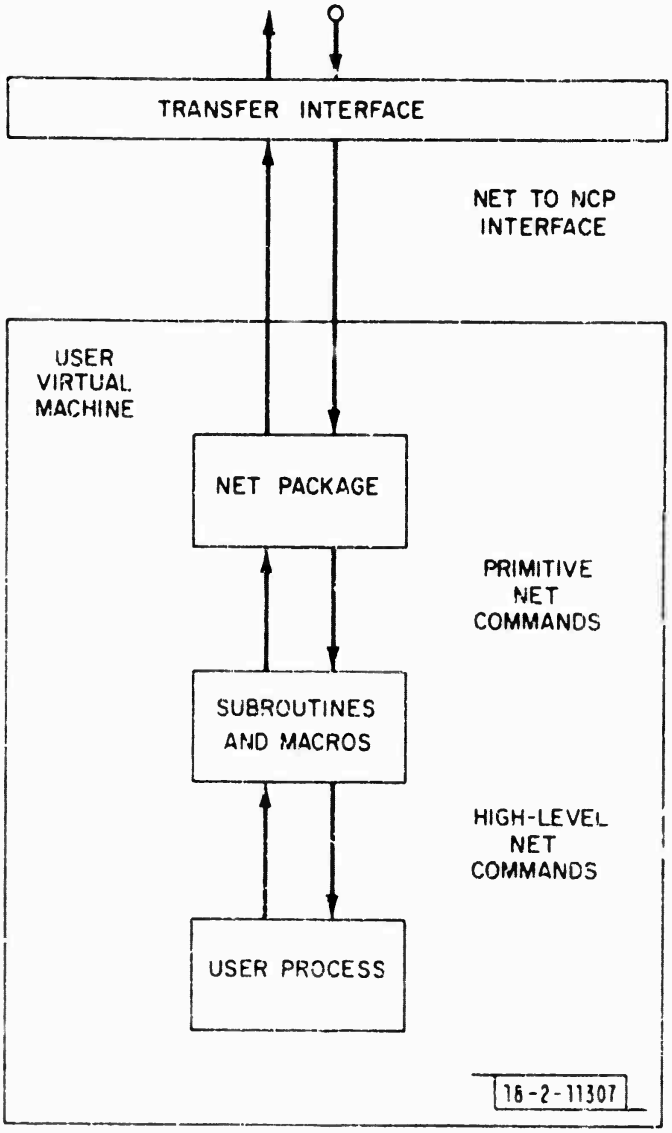


Fig. 8: NET Package

commands. These commands, which can be used directly by the user process, are interpreted by the NET package and issued as requests to the NCP through the medium of the VMCOM facility. Communications between NET and the NCP consist of an 80-byte control block together with a data block of from 0 to 1024 bytes.

The format of the control block is shown in Figs. 9, 10, and 11 as a set of field names, each followed by the field byte size in parentheses.

The CODE field, one byte, is used to specify the type of control block and the remaining fields are used for parameters set by either NET or the NCP. The CODE is given by a numeric value, in the range 0 to 12, and the significance of each is indicated in Fig. 12.

CODEs are defined which allowed NET to request the NCP to:

a. Issue, on its behalf, certain network commands such as STR (0), RTS (1), CLS (4), INS and INR (6).

b. Transmit, on its behalf, interprocess messages over established connections (2).

c. Deliver to it, interprocess messages from the network which the NCP had buffered until required (3).

d. Advise on the status of particular sockets (5).

e. Enable specified sockets to accept foreign STRs (9) or RTSs (10) without having to know in advance the foreign socket numbers. (The 'listen' function referred to in Section 2.6).

f. Disable sockets which previously had been enabled (11).

g. Purge all local references to connections not correctly closed by foreign HOSTs (12).

The NCP replies to each NET request with a reply control block (7) which contains appropriate status information in certain of its parameter fields. When the NET request is for the delivery of an interprocess message (3), the reply control block is followed by a data block, which represented the message. The NCP can also pass asynchronous interrupts to NET, at any time, by means of an interrupt control block (8). A number of interrupt codes are defined to notify the process of such events as:

a. The receipt of a network interrupt (INS or INR) on one of its current connections.

TN73-50-I(9)

```
NET TRANSMISSION CODE (1)
USER ID (8)_____________
LOCAL ID (3)              )     LOCAL SOCKET (4)
LOCAL TAG (1)_____________)
FOREIGN HOST (1)          )
FOREIGN ID (3)            )     FOREIGN SOCKET (5)
FOREIGN TAG (1)___________)
BIT COUNT (2)
INTERRUPT CODE (1)
STATUS (4)
BIT ALLOCATION LEFT (4)
MESSAGE ALLOCATION LEFT (2)
CONNECTION BYTE SIZE (1)
```

Fig. 9: NET to NCP Control Message Format

TN73-50-I(10)

```
CHARACTER CODE (4)________
LOCAL ID (3)              )     LOCAL SOCKET (4)
LOCAL TAG (1)_____________)
FOREIGN HOST (1)          )
FOREIGN ID (3)            )     FOREIGN SOCKET (5)
FOREIGN TAG (1)___________)
FLAG (1)
BIT COUNT SENT OR RECEIVED (2)
BUFFER ADDRESS (4)
STATUS (4)
BITS LEFT (4)
MESSAGES LEFT (2)
CONNECTION BYTE SIZE (1)
UNUSED BYTE (1)
ADDRESS OF INTERRUPT STACK (4)
```

Fig. 10: NET Command Format

TN73-50-I(11)

```
INTERRUPT CODE (1)
NEXT ENTRY IN INTERRUPT STACK (3)
LOCAL SOCKET (4)
FOREIGN SOCKET (5)
SPARE (3)
STATUS (4)
```

Fig. 11: Interrupt Stack Format

b. The acceptance, by a foreign HOST, of a request to establish a send connection to it, (i.e., the receipt of a foreign RTS which matched a previously issued STR).

c. The loading of NCP buffers for one of its receive connections.

d. The closing of one of its connections by a foreign HOST.

e. The matching of 'enabled' sockets with appropriate foreign STRs or RTSs.

f. The shutdown of the local NCP.

g. The resetting of any foreign NCP associated with one of its connections.

The interrupt codes are listed in Fig. 13 and any interrupt control block sent from the NCP to NET would contain one of these codes in the second byte of the BIT COUNT field, which doubles as the INTERRUPT CODE field (Fig. 9).

The USERID field it employed by both the NCP and NET to identify the virtual machine associated with the NET request. NET accesses the CP USERID, on behalf of its process, and completes this field accordingly. Sockets are defined (Section 2.2) to consist of an 8-bit HOST identifier concatenated with a 32-bit socket number. Local sockets are specified by the 32-bit socket number alone, since the local NCP can add its own HOST identifier as necessary. The LOCAL SOCKET field is divided into LOCAL ID and LOCAL TAG fields for convenience of mapping to process identifiers and ports, and the FOREIGN SOCKET field is similarly divided, but includes a FOREIGN HOST field. Both the LOCAL SOCKET and FOREIGN SOCKET fields are normally completed by NET, except in the case of enable requests where NET completes only the LOCAL SOCKET field and the NCP issues an interrupt control message, when the enabled socket is matched, with the FOREIGN SOCKET field completed. The BIT COUNT field is used by NET when sending an interprocess message to the NCP and by the NCP when delivering an interprocess message to NET.

The STATUS field is always completed by the NCP whenever it issues a reply or interrupt control message. The field is divided into 4 groups, each of 8 bits; the first two groups describe the reply status of the previous NET request, and the second two groups refer to the connection status of the associated entry in the RV table. Each bit, when set, indicates that a particular event has occurred or a particular state exists and the significance of each is given in Fig. 14.

TN73-50-I(12)

NET COMMAND CHARACTER CODE		NCP CONTROL MESSAGE CODE	DESCRIPTION
CON	(Connect)	0	Request NCP to issue an STR.
LIS	(Listen)	1	Request NCP to issue an RTS.
SND	(Send)	2	Request NCP to transmit a buffer.
RCV	(Receive)	3	Accept a buffer from the NCP.
CLS	(Close)	4	Request NCP to close a connection.
STA	(Status)	5	Request status from the NCP.
INT	(Interrupt)	6	Request NCP to issue an INS or INR.
NOP	(No-operation)	7	Request status from NET. Reply code from NCP.
-		8	Interrupt code from NCP.
ENC	(Enable connect)	9	Request NCP to accept a foreign STR.
ENL	(Enable listen)	10	Request NCP to accept a foreign RTS.
DIS	(Disable)	11	Request NCP to disable an enabled socket.
PUR	(Purge)	12	Request NCP to purge an entry.

Fig. 12: NET Commands

TN73-50-I(13)

INTERRUPT CODES	MEANING
0	Network interrupt received.
1	Foreign RTS received completing connection.
2	Receive buffers now loaded.
3	Foreign CLS received.
4	Enable connect now matched.
5	Enable listen now matched.
6	NCP has died.
7	Foreign NCP Reset (RST received).

Fig. 13: NCP to NET Interrupt Codes

Those bits marked with an asterisk (*) in the RV status are reset by the NCP after it has replied to an NCP status request.

The BITS LEFT and MESSAGES LEFT fields are completed by the NCP, whenever it replies to a NET request associated with a send connection, to notify NET of the current foreign allocation for that connection. The CONNECTION BYTE SIZE field is completed by NET for all connection requests, though only issued to the network, by the NCP, for send connections. A zero value on receive connection requests implies acceptance by NET of any connection byte size.

One advantage of this NCP mechanism is that a virtual machine, containing a process in communication with the network, can be 'logged off' without breaking network connections. All incoming messages for the process are buffered by the NCP and all information relating to interrupt control messages are recorded in the status flags. On resuming activity, the process can determine its current connections by requesting status from the NCP and can continue operations by taking the appropriate action on its connections.

Figure 10 shows the format of the parameter list to be used by a process when issuing a primitive NET command to the NET package. NET is accessed by a subroutine call which is associated with a pointer to the parameter list. The CHARACTER CODE field of the parameter list contains the mnemonic names of the command required and this is interpreted by NET which issues the appropriate control message to the NCP. The mnemonic names of all permitted commands are shown in Fig. 12 associated with the control message codes which are generated by NET. All commands cause the generation of a control message, with the single exception of NOP which is used to obtain the NET interrupt status. The LOCAL SOCKET, FOREIGN SOCKET, BIT COUNT, STATUS, BITS LEFT, MESSAGES LEFT and CONNECTION BYTE SIZE fields are all used for the same purposes as, and mapped by the NET package to, the fields of the same name in the control message. The FLAG field is reserved for use by the high-level NET commands and the BUFFER ADDRESS field contains a pointer either to a buffer holding the message for transmission in a SND command or to a buffer prepared for reception of a message in a RCV command.

NET maintains in free store an interrupt stack on behalf of its process. The ADDRESS OF INTERRUPT STACK field holds a pointer to the head of this stack, each entry of which is to be in the format shown in Fig. 11. If there are no entries, the pointer has a value of zero. NET will read interrupt control messages, from the NCP, either on receipt of a NOP command from the user process or when awaiting a reply control message from the NCP. Information contained in each interrupt control message is transferred to the appropriate fields of a separate

```
*
*         GROUP 1 - NET STATUS
*
OURIMP    EQU    X'80'    OUR IMP DEAD
ALLSTR    EQU    X'40'    ALL OUR FREE STORE IS IN USE
LNKTBL    EQU    X'20'    OUR IMPS LINK TABLE IS FULL
FORIMP    EQU    X'10'    FOREIGN IMP/HOST IS DEAD
FORLINK   EQU    X'08'    ALL LINKS TO HIM ARE IN USE
TAGMIS    EQU    X'04'    TAG IS OF WRONG POLARITY FOR RFC
RFCENB    EQU    X'02'    ATTEMPT TO ISSUE AN RFC ON ENABLED SOCKET
RFCRFC    EQU    X'01'    ATTEMPT TO REPEAT AN RFC
*
*         GROUP 2 - NET STATUS
*
SOCKMIS   EQU    X'80'    SOCKET MISMATCH
LINKMIS   EQU    X'40'    LINK IS OF WRONG POLARITY
TXREJ     EQU    X'20'    TRANSMISSION REJECTED - SEE RV STATUS
CMDREJ    EQU    X'10'    COMMAND REJECTED - SEE RV STATUS
DRAINON   EQU    X'08'    DRAIN SET ON
CLSCLS    EQU    X'04'    ATTEMPT TO ISSUE CLOSE ON CLOSED SOCKET
BADPAR    EQU    X'02'    BAD PARAMETERS
NOTPER    EQU    X'01'    NOT PERMITTED - HOST NOT KNOWN ETC
*
*         GROUP 3 - NET AND RV STATUS
*
OURCON    EQU    X'80'    OUR CONNECT/LISTEN ISSUED
HISCON    EQU    X'40'    HIS CONNECT/LISTEN ISSUED
BUFFLD    EQU    X'20'    BUFFERS ARE LOADED
INTRPT    EQU    X'10'    INTERRUPT RECEIVED                          *
OURCLS    EQU    X'08'    OUR CLOSE ISSUED
HISCLS    EQU    X'04'    HIS CLOSE ISSUED
CEASE     EQU    X'02'    CEASE ON LINK ISSUED
RFNM      EQU    X'01'    RFNM AWAITED
*
*         GROUP 4 - NET AND RV STATUS
*
ALLOCOK   EQU    X'80'    ALLOCATE SENT OR RECEIVED
ENABLED   EQU    X'40'    ENABLED FOR CONNECT OR LISTEN ONLY     **
ALLOCNOK  EQU    X'20'    ALLOCATE TOO SMALL FOR CURRENT ORDER ***
MSGFAIL   EQU    X'10'    LAST MESSAGE FAILED IN TRANSMISSION ****
SNDNOK    EQU    X'08'    SEND ORDER HAS BIT COUNT >8000       ****
BYMIS     EQU    X'04'    BYTE SIZE MISMATCH
UNUSED    EQU    X'02'
INTRPLST  EQU    X'01'    NETIN INTERRUPT LOST
*
*   (*) - THE INTRPT BIT IS CLEARED BY STATUS
*   (**) - THE ENABLED BIT IS CLEARED ON OUR LIS OR CON
*   (***) - THE ALLOCNOK BIT IS CLEARED WHEN MORE ALLOCATION IS RECEIVED
*   (****) - THE MSGFAIL AND SNDNOK BITS ARE CLEARED ON THE NEXT SND
*
```

Fig. 14: Format of Status Bits

entry in the interrupt stack, the entries being chained by means of the NEXT ENTRY IN INTERRUPT STACK field. The last entry contains a value of zero in this field. It is the responsibility of the user process (or the high-level NET commands) to take action on the stack entries, to return to free storage the block associated with each one and to maintain the pointer in the ADDRESS OF INTERRUPT STACK field.

In addition to the full status reply in the STATUS field of the parameter list, a return code is placed in general purpose register 15 in order to conform with the standard CMS interface. This, together with the standard calling procedure, allows the NET package to be used by a process written in any of the languages supported by the CMS system. See Appendix G.

It is expected that most processes would access NET through the set of macros which effect the high-level NET commands (see Appendix H). For each connection required, a NETSOCK or NETSOCKV macro is to be specified which generates a buffer for the NET command parameter list (Fig. 10) and associates identifiers with certain of its fields. Before opening a connection a NETID macro is to be issued, which initializes the FLAG field of the parameter list and obtains a value for the LOCAL ID field of the LOCAL SOCKET. The CP UTABLE address for the virtual machine is used for the ID field. A NETOPEN macro can then be issued to establish the connection whose polarity is determined by the value in the LOCAL TAG field. This can be followed by NETREAD or NETWRITE macros, as appropriate to the polarity, in order to transfer interprocess messages. Flags (Fig. 15) are set in the FLAG field of the appropriate parameter list, associated with each connection, by a routine which analyzes and collapses the interrupt stack each time a high-level command is issued. A NETNOP macro is also provided for this purpose. Network interrupts can be issued by means of the NETINT macro, status obtained by means of NETSTAT and the connection closed by means of NETCLOSE.

Most macros expand into code which calls subroutines of the same name as the macro. Each subroutine is associated with an appropriate parameter list and generates one or more calls to the NET package.

The arrival of interrupt control messages, from the NCP, is detected by an interrupt from the VMCOM device on the virtual machine. A process, therefore, which wishes to suspend itself awaiting some network activity can issue a CMS WAIT SVC on the VMCOM device until an interrupt control message has arrived. A NETWAIT macro is provided for this purpose. When the interrupt arrives and control is returned from NETWAIT, the process can then issue a NETNOP macro and act on its flags accordingly. An example of this procedure, together with the use of most of the macros, is given in the TELNET process.

```
*
*        EQUS FOR FLAGS IN NETPLIST
*
OPEN     EQU   X'80'          OPEN/NOTOPEN
SEND     EQU   X'40'          SEND/RECEIVE
NETIN    EQU   X'20'          NETWORK INTERRUPT
LISTEN   EQU   X'10'          LISTEN ISSUED
BUFFERS  EQU   X'08'          BUFFERS LOADED
CLOSED   EQU   X'04'          LINK CLOSED
ENABLE   EQU   X'02'          ENABLE MATCHED
DEAD     EQU   X'01'          NCP DIED OR RESET ISSUED
*
```

OPEN	Set when OPEN or OPENE macro successfully issued Cleared on successful close.
SEND	Set when OPEN or OPENE macro issued on send socket Set when enable on send socket issued Cleared on successful close.
NETIN	Set on receipt on network interrupt Cleared on a netstat before STA issued.
LISTEN	Set on receipt on a LISTEN interrupt Cleared on a netstat before STA issued.
BUFFERS	Set on receipt of a Buffers loaded/ready interrupt Cleared on a netread before RCV issued Cleared on a netwrite before SND issued Cleared on a netstat before STA issued.
CLOSED	Set on receipt of a socket closed interrupt Cleared on successful net close.
ENABLED	Set on receipt on an enabled completed interrupt Cleared on a netopen.
DEAD	Set on receipt of an NCP dead interrupt Not reset.

Fig. 15: NET Flag Bits

4.3 The Logger-Server

The NET system is designed to operate in the CMS environment which supports a single user process only. When more than one user process is required to be concurrently active in a virtual machine, as in the case of the logger-server, a polling or scheduling routine is needed and the high-level NET commands cannot be used. One reason for this is because the high-level commands deny access to the interrupt stack maintained by NET and for multi-process working it is necessary for the polling routine to associate NET interrupts with the separate processes. Rather than complicate the high-level command interface, it was decided to use the primitive NET commands for the logger-server system.

The logger-server is designed to be interrupt driven by a simple polling routine which activates a number of processes in turn and suspends itself whenever all outstanding interrupts have been serviced. Four devices can raise interrupts on the logger-server virtual machine and thus cause the polling sequence to be resumed. These are the VMCOM device, whose interrupt signifies the receipt of an interrupt control message from the NCP, and three CP virtual terminals whose interrupts indicated the presence of messages for their associated remote users. Associated with each process is a table which contains sufficient details on the state of the process to enable it to be restarted from its current suspension point. The table includes copies of the program status word (PSW), all general purpose registers, buffers, flags and pointers (see Fig. 16).

The poller accesses two flags in each table; the WAIT flag and the START flag. If the WAIT flag of a process is set, that process will not be re-activated during the current poll, but if it is clear, the poller will set it and test the state of the START flag. If the START flag is clear, the poller will restore all registers from the table and re-activate the process at the point defined by its PSW, but if it is set, the poller will clear it and activate the process at a start address specified by a start entry in the table.

Four processes are supported by the poller; an enabling process and three CP virtual terminal control processes. The control processes are all identical in function and are implemented by a single segment of re-entrant code. An interrupt handler is implemented which recognizes all interrupts on the virtual machine and associates each of the four devices with a particular process. The VMCOM device is associated with the enabling routine and each of the CP virtual terminals is associated with one of the control processes. On receipt of an interrupt, the interrupt handler examines the START flag of the associated process and if this is clear it clears the WAIT flag. If, however, the START flag is set, it ignores the interrupt. Since a device interrupt will

```
*
WT1         EQU   X'80'   WAIT FLAG FOR POLLER
DE1         EQU   X'40'   DEVICE END
UE1         EQU   X'20'   UNIT EXCEPTION
UC1         EQU   X'10'   UNIT CHECK
AT1         EQU   X'08'   ATTENTION
UN1         EQU   X'04'   UNEXPECTED INTERRUPTS
ST1         EQU   X'02'   START FLAG - USED FOR ENABLING
IN1         EQU   X'01'   INTERRUPT STACK FLAG
*
HI2         EQU   X'80'   HIO OUTSTANDING
MO2         EQU   X'40'   MORE TC COME FROM NCP BUFFERS
AW2         EQU   X'20'   AWAITING DESPATCH TO CP
ADDCR       EQU   X'10'   CARRIAGE RETURN TO BE ADDED
CR          EQU   X'08'   CARRIAGE RETURN DETECTED AS LAST IN BUF
BUFPURGE    EQU   X'04'   BUFFER PURGE IN PROGRESS
HINT        EQU   X'02'   HIO INTERRUPT AWAITED
REPLOG      EQU   X'01'   REPEAT LOGIN FLAG
*
INTS        EQU   X'80'   INTERRUPT ON SEND LINK
LISS        EQU   X'40'   LISTEN ON SEND LINK
BUFS        EQU   X'20'   BUFFERS LOADED ON SEND LINK
CLSS        EQU   X'10'   CLOSE ON SEND LINK
ENCS        EQU   X'08'   ENABLE CONNECT ON SEND LINK
ENLS        EQU   X'04'   ENABLE LISTEN ON SEND LINK
NCPS        EQU   X'02'   NCP DIED ON SEND LINK
CHKPW       EQU   X'01'   CHECK PASSWORD FLAG
*
INTR        EQU   X'80'   INTERRUPT ON RCV LINK
LISR        EQU   X'40'   LISTEN ON RCV LINK
BUFR        EQU   X'20'   BUFFERS LOADED ON RCV LINK
CLSR        EQU   X'10'   CLOSE ON RCV LINK
ENCR        EQU   X'08'   ENABLE CONNECT ON RCV LINK
ENLR        EQU   X'04'   ENABLE LISTEN ON RCV LINK
NCPR        EQU   X'02'   NCP DIED ON RCV LINK
DOWN        EQU   X'01'   LOGGER GOING DOWN
*
```

Fig. 16: LOGGER Routine Flags

also re-start the poller, if it is suspended, the process which services the device will eventually be re-activated.

In the case of VMCOM interrupts, which signifies the arrival of an interrupt control message from the NCP, the enabling routine issues a NOP command to NET followed by a subroutine call to an interrupt stack handler. This handler analyzes the NET interrupt stack and generates from it separate interrupt stacks for each of the processes. When a new entry for a particular process is received, the WAIT flag for that process is cleared and its INTERRUPT STACK flag set.

A process suspends itself, when awaiting network or device activity, by means of a special wait subroutine (WBACK) which saves all process registers and the current PSW in the appropriate table before returning control to the poller.

On first becoming established, the logger-server calls an initializing subroutine which attempts to close all connections between itself and the network, purge all references to any of its sockets from the NCP, and log out all CP virtual terminals. Following this, the process tables are initialized and the WAIT and START flags set for each process except the enabling routine, for which only the START flag is set. Control is then passed to the poller.

The poller can activate only the enabling routine at this time, and this causes an enable connect (ENC) command to be issued to NET for the initial connection socket (ICS) of the logger-server. The routine then suspends itself by means of the WBACK subroutine.

When a network user completes a connection to the ICS of the logger-server, the NCP sends an appropriate interrupt control message. This causes the enabling routine to be re-activated by the mechanisms outlined above. On re-activation, this routine accepts the request for connection from the network user if there is at least one CP virtual terminal unassigned. It will issue the appropriate NET commands, in accordance with the Initial Connection Protocol (ICP) described in Section 2.6, to establish duplex communications between the sockets of the next unassigned CP virtual terminal process and the network user process. On the successful completion of this protocol, the enabling routine clears the WAIT bit associated with the newly assigned control process, issues an enable connect for the initial connection socket, and suspends itself, ready to repeat the cycle of events. If there is no control process unassigned, the enabling routine refuses the request for connection, and then issues the enable connect command followed by the WBACK subroutine call.

The newly assigned control process will, in its turn, be activated by the poller at its start address and will commence to transfer information between the network user process and the CP virtual terminal interface, in accordance with the TELNET protocol and the CP terminal conventions. Initially, the control process is concerned with the logging on sequence, the verification that the required virtual machine is marked for network use, and the mapping of a network password to the local one. Subsequent activity is concerned with the actual passage of messages between the network user process and its local virtual machine; the translation of characters, as required, between ASCII and EBCDIC; and the interpretation of special control characters such as 'break', 'reverse break' and 'synchronize'.

Whenever network or device responses are required, the process suspends itself, using the WBACK subroutine, until the response has been received. During this period, one of the other processes can be re-activated and thus the poller is able to support the enabling process and the three CP virtual terminal processes concurrently, without any apparent loss in response.

The system is implemented in three program segments; LOGMAIN, LOGDEV, and LOGST. LOGMAIN contains a small initializing routine, the poller, the enabling routine, the interrupt handler, the NET interrupt stack handler and various subroutines for stack analysis and character code conversion. LOGDEV is the re-entrant segment which effects the three CP virtual terminal control processes and LOGST is the main initializing routiine.

The logger-server can be closed down by issuing to its virtual machine an 'external' interrupt. This causes the WAIT flag to be cleared for all processes in use and the DOWN flag to be set. On re-activation, each CP virtual terminal control process will issue a 'LOGGER CLOSING DOWN' message to its network user and then close its network connections. When all processes have terminated, the logger-server halts.

Error recovery is directed at maintaining a service to the network as a whole, for as long as possible. Errors detected in CP virtual terminal control processes are either network command rejections by the NCP, through NET, or virtual terminal transmission rejections by CP. In either case, the recovery of the failing process is not attempted but the virtual machine associated with it is logged off, its network connections are closed, and its process table re-initialized. In this way, the integrity of the other processes are maintained, the network user is informed of the failure by the closing of his connections and the facilities of the failing process are recovered for re-use. Errors detected by the enabling process are network command rejections only. Certain rejections are recoverable as in the case of an initial SND command failing because of an

insufficient allocation from the network user. In such cases the command is repeated, at a reasonable frequency, until it is either accepted or a repeat count has become exhausted. In the latter situation, the connection is then closed. Other rejections are not recoverable, as in the case of an enable connect command failing. Under these circumstances, the logger-server is shut down after a trace flag has been set, indicating the source of the error, and a core dump output to the line printer. This procedure is also followed if a 'program' interrupt condition, signifying a fault in the program code, is signalled by CP.

4.4 The Network Control Program

As in the case of the logger-server, the network control program (NCP) is designed to be interrupt driven by a simple poller which supports a number of concurrent processes in the NCP virtual machine. An outline, in block schematic form, of the inter-relationships between these processes is shown in Fig. 17.

Each process is implemented by a separate segment of code and is identified on the diagram by its segment name.

Communication with user processes, in other virtual machines, is through the VMCOM interface which is described in Section 4.2. Communication with the IMP is through the IMI hardware interface, which was designed by BRYAN (Ref. 3) and is described in Appendix 12.

NCPINIT performs initialization and error recovery tasks and NCPMAIN contains the poller, the interrupt handler and various subroutines associated with accounting and recording activities. Processes are re-activated according to the state of flags in the PORT TABLE which are updated by other processes and the interrupt handler.

NCPXFER accepts control messages from the VMCOM interface and interprets them in accordance with the NET to NCP control message protocol (Fig. 9). Depending upon the type of request and the current state of the RV TABLE, NCPXFER adds items to the IMP'S QUEUE for transmission over the network, adjusts entries in the RV TABLE and issues a reply to the user process via the VMCOM interface.

NCPIMPO is concerned solely with the output of the IMP'S QUEUE to the network and the associated control of the IMP hardware interface. A facility is included whereby this output can be channeled directly into the input buffers of NCPIMPI, for test purposes or for local use when the IMP is not available.

NCPIMPI processes all input from the IMP and implements the HOST-to-IMP protocol. The HOST-to-HOST protocol is

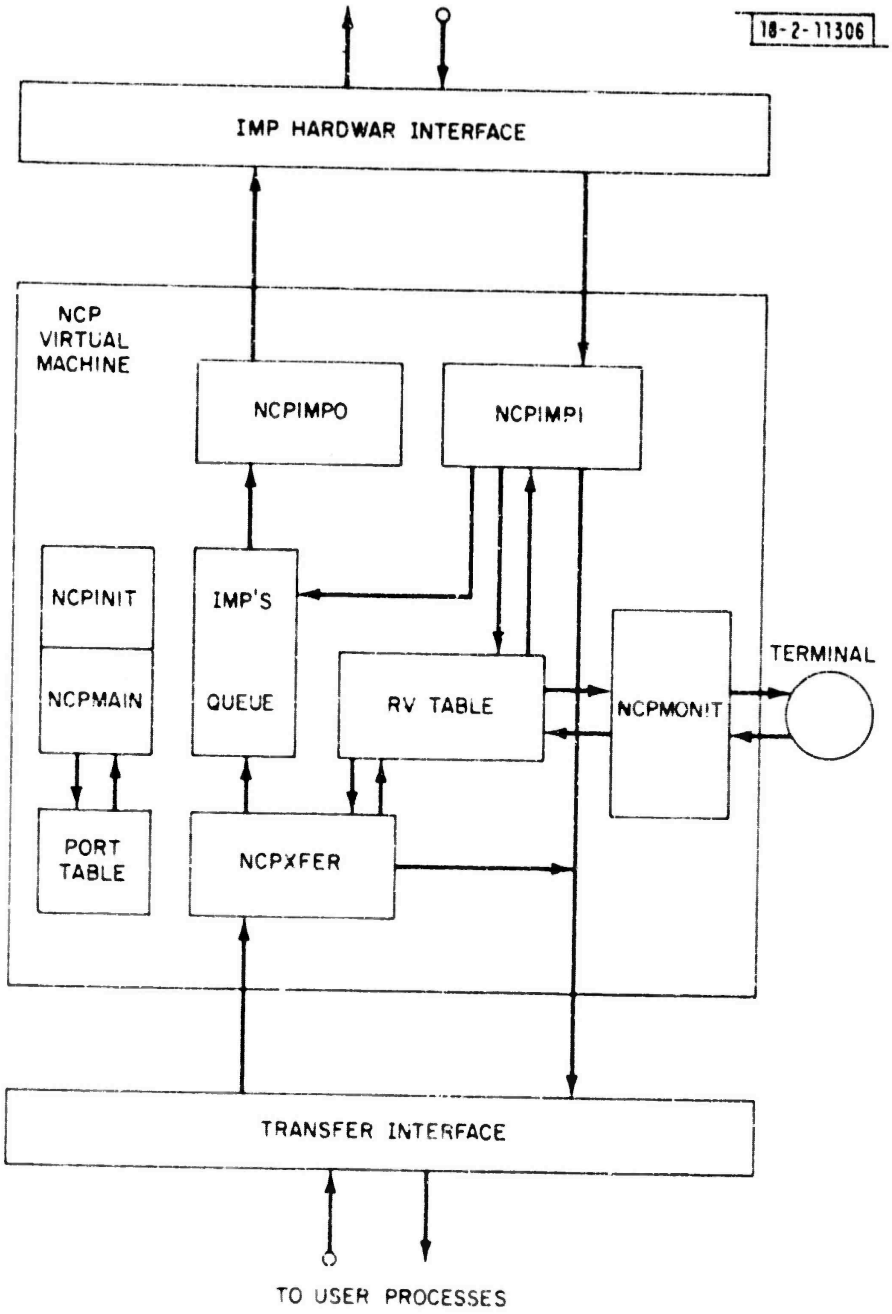


Fig. 17: The Network Control Program

implemented partly by NCPIMPI and partly by NCPXFER. Depending upon the type of message from the network and the current state of the RV TABLE, NCPIMPI loads buffers for collection by local user processes, sends interrupt control messages over the VMCOM interface, adjusts entries in the RV TABLE and add items to the IMP'S QUEUE in reply to network messages.

NCPMONIT provides facilities whereby entries in the RV TABLE can be inspected, certain network commands generated, NCP status monitored and the system cleanly closed down.

4.4.1 NCPMAIN

The polling mechanism of NCPMAIN is similar to that of the logger-server. Interrupts received from external devices are flagged in a PORT TABLE which is used to determine the next process for re-activation. Each device is associated with a process through an entry in this table. Entries associate the input channel from the IMP with NCPIMPI; the output channel to the IMP with NCPIMPO; the VMCOM interface with NCPXFER; and the console terminal with NCPMONIT. No entry is required for the output portion of the VMCOM interface.

Each entry in the PORT TABLE has the format shown in Fig. 18 which was originally designed for duplex devices.

It was subsequently found more convenient to implement simplex entries but the earlier structure was retained. The format is described as a set of field names, each followed by its field byte size in parentheses. Two sets of flags are defined; POLLING FLAGS which are used by the poller to determine whether the process should be reactivated, and if so, at which address; and DEVICE FLAGS which indicate the current state of the associated device. The significance of both sets of flags is given in Fig. 19.

Other fields contain the start address of the process, save areas for its PSW and registers, and a buffer address. These fields are duplicated, in each entry, to provide facilities for both input and output channels. The CMS DEVICE ADDRESS and DEVICE ID are also specified in the PORT TABLE entry.

The format of the buffers is given in Fig. 20, and provides fields for a variable length MAIN BUFFER, the current CHANNEL STATUS WORD (CSW) of the device, and workspace for the process.

On receipt of an interrupt from a device, the interrupt handler sets the associated DEVICE FLAGS, copies the CSW to the appropriate field in the buffer, and clears the WAITSET bit in the POLLING FLAGS. The MAIN BUFFER is used by the device for any input or output transfers.

TN73-50-I(18)

```
POLLING FLAGS (1)
DEVICE INPUT ROUTINE ADDRESS (3)
DEVICE INPUT ROUTINE PSW (8)
DEVICE INPUT ROUTINE REGISTERS (64)
INPUT BUFFER ADDRESS (4)
DEVICE FLAGS (1)
DEVICE OUTPUT ROUTINE ADDRESS (3)
DEVICE OUTPUT ROUTINE PSW (8)
DEVICE OUTPUT ROUTINE REGISTERS (64)
OUTPUT BUFFER ADDRESS (4)
SPARE (2)
DEVICE ADDRESS (2)
DEVICE ID (4)
```

Fig. 18: PORT TABLE FORMAT

TN73-50-I(19)

```
     POLLING FLAGS

BUFIN     EQU X'80' INPUT BUFFER LOADED
BUFOUT    EQU X'40' OUTPUT BUFFER LOADED
STATIN    EQU X'20' INPUT STATE BUSY
STATOUT   EQU X'10' OUTPUT STATE BUSY
WAITSET   EQU X'08' WAIT STATE SET
SNDSET    EQU X'04' SEND STATE _ NOT RECEIVE

     DEVICE FLAGS

INPROG    EQU X'80' IN PROGRESS STATE
DEVEND    EQU X'40' DEVICE END
UEX       EQU X'20' UNIT EXCEPTION
UCHK      EQU X'10' UNIT CHECK
ATTN      EQU X'08' ATTENTION
UNEXBIT   EQU X'04' UNEXPECTED BITS
```

Fig. 19: Polling and Device Flags

TN73-50-I(20)

```
POINTER TO MAIN BUFFER (4)
SIZE OF MAIN BUFFER (4)
CHANNEL STATUS WORD (4)
WORKSPACE (variable)
MAIN BUFFER (variable)
```

Fig. 20: Input and Output Buffer Formats

The poller, if suspended on a CMS WAIT, is restarted on the receipt of an interrupt and re-activates the first process whose WAITSET bit is clear. The re-activation is either at the start address of the process or at its current PSW depending upon the state of its STATIN or STATOUT flags. Additionally, an output process requires its BUFOUT flag to be set before the poller re-activates it.

After servicing the data buffer and any current interrupts associated with a device, the process saves its registers and PSW in the appropriate fields and returns control to the poller, awaiting further interrupts. Process activity typically includes the initiation of further transfers over the device channel and the setting of certain POLLING FLAGS in the PORT TABLE entries of other processes.

NCPMAIN contains a subroutine for writing accounting records to disk, in the format shown in Fig. 21.

The subroutine is called by NCPXFER and NCPIMPI to write a record each time any connection becomes fully established or fully closed. The records are completed from the appropriate entry in the RV TABLE and include the time at which the first Request for Connection (RFC) is made; the time at which the connection became fully established; the time at which the first close is received; and the time at which the connection became fully closed. Other fields in the record contain the total number of messages and bits issued over the connection; the local USERID; the identities of both sockets; the foreign HOST and the link number. Although charges are not being raised against network users it is considered desirable that sufficient information be recorded to allow the identification of all users and their utilization of system resources.

Other elements contained in NCPMAIN includes a subroutine for writing logging records to disk and a HOSTS WE KNOW table. The subroutine is used for recording any unusual events or error conditions detected by the NCP. The HOSTS WE KNOW table is used to maintain an entry for the current status and link utilization of every HOST on the network.

4.4.2 NCPXFER

NCPXFER consists of a routine to read messages from the VMCOM interface and a set of command routines. Each message is analyzed and interpreted as a command from NET and the appropriate command routine is then entered. These routines typically generate, refer to, and modify entries in the RV TABLE; add network messages to the IMP's QUEUE, and send reply messages back to the requesting local process.

TN73-50-I(21)

```
USERID (8)
DATE WRITTEN (8)
FIRST RFC TIME (8)
COMPLETED CONNECTION TIME (8)
FIRST CLOSE TIME (8)
COMPLETED CLOSE TIME (8)
LOCAL SOCKET (4)
FOREIGN SOCKET (4)
FOREIGN HOST (1)
FOREIGN LINK (1)
RV TABLE STATUS (2)
NUMBER OF MESSAGES (4)
NUMBER OF BITS (4)
```

Fig. 21: Accounting Record Format

TN73-50-I(22)

```
USERID (8)_____________
LOCAL ID (3)            )    LOCAL SOCKET (4)
LOCAL TAG (1)__________)
FOREIGN HOST (1)        )
FOREIGN ID (3)          )    FOREIGN SOCKET (5)
FOREIGN TAG (1)________)
LINK (1)
STATUS (2)
CONNECTION BYTE SIZE (1)
HEAD OF BUFFER CHAIN (3)
TEST BYTE (1)
TAIL OF BUFFER CHAIN (3)
ADDRESS OF NEXT RV ENTRY (4)
MESSAGE ALLOCATION (2)
MESSAGE ALLOCATION (2)
MESSAGE FOR RETURN (2)
DATE (8)
TIME1 (8)
TIME2 (8)
NUMBER OF MESSAGES TRANSMITTED OVER CONNECTION (4)
NUMBER OF BITS TRANSMITTED OVER CONNECTION (4)
BITS FOR RETURN (4)
```

Fig. 22: RV Table Format

The RV TABLE contains an entry for every connection supported by the NCP and the format for each entry is shown in Fig. 22.

Fields are provided for the USERID of the local process; the local and foreign sockets; the link number assigned to the connection; the connection byte size and the current message and bit allocations. Accounting fields contain details of the date; the time of connecting; the time of closing and the cumulative number of messages and bits transmitted over the connection. The STATUS field holds a set of flags (Fig. 14), describing the current state of the connection and address fields point to the head and tail of a buffer chain associated with it and the next entry in the table. The MESSAGES FOR RETURN and BITS FOR RETURN fields are used to determine when a new allocate command should be issued to a foreign NCP.

RV TABLE entries are established for control connections, on links zero, to all HOSTs in the HOSTS WE KNOW table, during the initialization phase. This enables a common, internal procedure to be adopted for all network transmissions. Any message for output to the network, including HOST-to-HOST control messages, is formatted by the originating routine and added to the chain of buffers supported by the appropriate send connection entry in the RV TABLE. A pointer to this RV TABLE entry is then added to the IMP'S QUEUE, unless one already exists. HOST-to-IMP messages are treated as a special case and are only issued during initialization and shutdown by NCPINIT. NCPIMPO uses the pointers in the IMP'S QUEUE to access the RV TABLE entry and the associated buffers for transmission, controlling flow over the connection by means of the RFNM AWAITED flag.

Messages received from the network are analyzed by NCPIMPI, and all IMP-to-HOST and HOST-to-HOST messages are trapped and acted upon directly by this routine. Process-to-process messages are stripped of their message headers and added to the buffer chain of the appropriate receive connection entry in the RV TABLE.

The CONNECT and LISTEN command routines of NCPXFER, after validating the request, examine the RV TABLE for a complementary entry set by NCPIMPI from the network, and, if none exists, generate a new one. An STR or RTS control message will, in either case, be added to the buffer chain of the appropriate link zero send entry, for transmission to the foreign NCP. When a LISTEN request matches a foreign STR entry, an initial ALL control message is also added to the same buffer chain. The SEND command routine examines the RV TABLE for the appropriate send connection entry and, after ensuring that there is sufficient allocation, adds the interprocess message to its buffer chain, decrementing the allocation fields accordingly. The RECEIVE command routine, after having found the appropriate receive

connection entry, collects the next buffer from the buffer chain and passes it to the requesting local process. It increments the MESSAGES FOR RETURN and BITS FOR RETURN fields by the size of the message and if either value exceeds a given thresholds limit, it chains an ALL control message to the associated link zero send entry buffer chain and resets the return fields.

The other command routines follow similar sequences; the CLOSE routine issues a CLS control message and removes the RV TABLE entry if the foreign CLS has already been received; the ENABLE CONNECT and ENABLE LISTEN routines generate new RV TABLE entries without issuing network messages; the DISABLE and PURGE routines remove entries without network messages; the INTERRUPT routine issues an INS or INR control message; and the STATUS routine clears certain bits in the STATUS field after placing its value, together with the system status, in the reply control message buffer.

In all cases the command routine issues a reply control message to the requesting local process and returns control to the calling routine. This routine returns control to the poller, with its WAITSET flag clear if there are further messages to be read, or with it set if the last message has been analyzed.

4.4.3 NCPIMPI

This process accepts and anlyzes input information from either the real IMP hardware interface or from an artificial software 'IMP'. NCPINIT activates, during the initialization phase, either the hardware device control routines in both NCPIMPO and NCPIMPI, or the corresponding artificial IMP routines, in accordance with a load time parameter. The NCPIMP artificial IMP routine passes the output from the NCP directly to the analysis routine of NCPIMPI via an internal buffer. This mode of operation is to test new sections of the NCP without disturbing the rest of the network.

The same analysis routine is used in either case, and separates input into three classes of messages; irregular messages of the IMP-to-HOST PROTOCOL; regular messages on link zero of the HOST-to-HOST protocol; and regular messages on non-zero links which are part of interprocess transmissions.

Most irregular messages are logged, by means of a subroutine in NCPMAIN, and appropriate flags are adjusted in RV TABLE ENTRIES, HOSTS WE KNOW status fields, and system status words. A type 5 irregular message (RFNM) causes the RFNM AWAITED flag to be cleared in the associated RV TABLE entry and the NCPIMPO routine to be flagged for re-activation. A type 7 (destination IMP or HOST dead) causes a flag to be set in a particular entry of the HOSTS WE KNOW table, the

RFNM AWAITED flag to be cleared in an RV TABLE entry, and a logging record to be written to disk. Other irregular messages cause similar sequences to be followed.

Host-to-Host control messages, on links zero, are interpreted as one or more control commands and cause the appropriate control command routine to be entered. These routines perform similar tasks and are complementary to the command routines of NCPXFER. The STR and RTS control command routines, after validating the format of the command, examine the RV TABLE for a matching entry set by NCPXFER on behalf of a local process, and, if none exists, generate a new one. When an STR matches a local LISTEN REQUEST, an initial ALL control message is added to the buffer chain of the appropriate link zero send entry in the RV TABLE. When an STR matches an ENABLE LISTEN or an RTS matches an ENABLE CONNECT, an interrupt control message is sent, over the VMCOM interface, to the associated process. The CLS routine clears the RV TABLE entry if a CLOSE command has already been issued by the local process, otherwise, it chains a CLS control command to the appropriate link zero entry in the RV TABLE and issues an interrupt control messages to the local process. INS and INR command routines issue interrupt control messages to the local process; the GVB routine returns the required fraction of the allocation by chaining a RET control command to a link zero entry in the RV TABLE; and RST removes all RV TABLE entries associated with the issuing NCP and warns all local processes involved via an interrupt control message. Similar sequences are followed by the other command routines.

If errors are detected in a message from a foreign host, an ERR control message is returned to that host. Errors that may occur are:

a. Illegal OP Code encountered in operations deblocking.

b. Short parameter space. End of message encountered before all expected parameters.

c. Bad parameter(s), e.g., two receive (send) sockets in a RTS (STR), link number not 1<L<32, bad socket polarity within command.

d. Request on a closed (null) socket. A request (other than RTS/STR) was made for a non-existent socket.

e. Socket (link) not connected. A request which requires a connected socket (link), (i.e., ALL, INR, transmit) was made for an existing but not connected socket (link).

f. Byte size not consistent.

g. RFC for your socket already issued by me (timing error).

h. RFC on an existing socket pair.

i. RFC on a closed socket pair.

j. Allocation too small for this transmission.

Only the first five errors have been given a network defined error code. Refer to Appendix D for a description of the ERR control message.

Regular messages on non-zero links are stripped of their message headers and added to the buffer chain of the associated receive entry in the RV TABLE. If the BUFFLD flag (Fig. 14) is clear, it is set and an interrupt control message sent to the local user process.

After completing all activity associated with the current input buffer, NCPIMPI issues a new input transfer order to the IMP interface and suspends itself, to await further interrupts, by returning control to the poller.

4.4.4 NCPIMPO

NCPIMPO accesses buffers indirectly pointed to by entries in the IMP'S QUEUE and outputs their contents to either the real IMP hardware interface or the artificial IMP internal buffer. Only messages addressed to (and originated by) the local NCP are passed by the artificial IMP, since when operation is in this mode, all foreign HOSTs are flagged as being dead in the HOSTS WE KNOW table.

For each entry in the IMP'S QUEUE, the RV TABLE entry is accessed and if the connection has not been closed and the RFNM AWAITED flag is not set, the next buffer in the buffer chain is output to the IMP, and the RFNM AWAITED flag set. As buffers are output, the routine frees the associated storage and when all buffers in a chain have been output the corresponding pointer in the IMP'S QUEUE is removed.

When a connection is fully closed, the closing routine (NCPXFER or NCPIMPI) frees any storage associated with the buffers in the buffer chain, removes the entry from the RV TABLE and flags the pointer in the IMP'S QUEUE as 'closed'. NCPIMPO, on detecting such a pointer, removes it from the IMP'S QUEUE.

After initiating a transfer, NCPIMPO suspends itself by returning control to the poller with its WAITSET and BUFOUT flags set, awaiting reply interrupts from the IMP. On re-activation, after completion of the transfer, NCPIMPO attempts to output further buffers, until either all have been tranferred or only buffers associated with

RFNM AWAITED flags are left. Under these circumstances, it suspends itself, by returning control to the poller with its WAITSET and BUFOUT flags clear. It can then only be re-activated by NCPIMPI or NCPXFER setting its BUFOUT flag. This occurs when NCPIMPI clears a RFNM AWAITED flag in the RV TABLE as a result of a network message, or when NCPIMPI or NCPXFER adds a new entry to the IMP'S QUEUE

4.4.5 NCPMONIT

The NCP virtual machine can be run disconnected from its control console which is the normal mode of operation employed. For monitor purposes, however, the virtual machine is re-connected, usually to a visual display unit, and the NCPMONIT process activated by means of an 'external' interrupt. This interrupt is detected by a routine in NCPINIT which cause the typing of a monitor message and the clearing of the WAITSET flag in the NCPMCNIT entry of the PORT TABLE.

Once activated, NCPMONIT attempts to read and obey monitor commands typed on the console keyboard until a BEGIN command is entered. This command de-activates the NCPMONIT process until the next 'external' interrupt is received.

In addition to BEGIN, commands are available to perform the following functions:

a. Disconnect the console from the NCP virtual machine and obey a BEGIN command (DISC).

b. Set the DRAIN flag in the system status word to deny any further requests for connection (DRAIN). This is normally issued shortly before shutdown.

c. Issue an echo network control command to a named foreign HOST, await the echo reply, check its validity and type an appropriate message on the console (ECHO). This is useful for determining whether the NCP or a remote HOST is active or not.

d. Output to the console, the current status of all HOSTS in the HOSTS WE KNOW table (FHOST).

f. Issue a NOP network control command to all HOSTs in the HOSTS WE KNOW table and obey a DISC MONITOR COMMAND (NOP). This is useful in that it obtains, from the network replies, the current status of all HOSTs.

g. Issue an RST network control command to all HOSTs in the HOSTS WE KNOW TABLE AND OBEY A DISC monitor command (RESET) This is issued whenever the NCP is re-established after a shutdown.

h. List, either on the console or to a named disk

file, all the entries in the RV TABLE (RV).

i. List, on the console, all the entries in the RV TABLE associated with a named socket (SOCKET).

j. Output to, either the console or a named disk file, a record of the current utilization of the NCP (STATUS).

k. Shut the NCP down cleanly by closing all current connections, notifying all local users and warning the IMP (SHUTDOWN). The final part of this process is performed by NCPINIT.

l. List, on the console, all the entries in the RV TABLE associated with a named local virtual machine (USER).

4.4.6 NCPINIT and Error Recovery

NCPINIT consisted of a set of routines which are called by NCPMAIN for initialization, by NCPMONIT for shutdown, and by all processes for error recovery.

The initialization routine reads and analyzes load time parameters which determine whether the real or artificial IMP routines are to be established in NCPIMPI and NCPIMPO, and whether or not NCPMONIT is to be activated at load time. Additional parameters can be 'stacked' in the console input buffer by this routine, for later interpretation, by NCPMONIT. This mechanism is normally used to issue a RESET monitor command when the system first becomes active.

After analysis of the load time parameters, the interrupt handler in NCPMAIN is established and all control link entries are set in the RV TABLE. Selection of the appropriate device control routines in NCPIMPI and NCPIMPO is then made and if the artificial IMP has been specified, all foreign HOSTs are flagged as dead in the HOSTS WE KNOW TABLE. If the real IMP is specified, the hardware interface is activated and HOST-to-IMP protocol NOP messages are sent to the IMP until a satisfactory reply has been received. A HOST-to-HOST PROTOCOL ECO control command is then sent, by NCPINIT, to itself, and when this is received correctly the IMP interface is considered to be established.

Before returning control to NCPMAIN, the program interrupt PSW is set to the address of the error recovery routine in NCPINIT and the external interrupt trap to the address of the external interrupt handler.

When an external interrupt is received, the routine in NCPINIT determines whether or not a console is connected to the NCP virtual machine and, if one is connected, activates NCPMONIT. If no console is connected, a SHUTDOWN monitor message is stacked in the console input

buffer before the activation of NCPMONIT. The system can, therefore, be closed down simply by sending an external interrupt to the disconnected NCP virtual machine.

The shutdown routine in NCPINIT issues an interrupt control message to all local users, a HOST-to-IMP protocol 'HOST going down' message to the IMP, and closes the hardware interface. It then clears the interrupt handler and returns control to CMS with a return code set. When the shutdown is a direct result of a SHUTDOWN monitor command, all network connections are closed by NCPMOINT in the normal way, and control returned to CMS with a return code of zero.

In the case of an involuntary crash, caused by a program interrupt from CP, the program interrupt PSW is set to the address of a final error recovery routine; a copy of all registers and core storage is dumped to the line printer; the event logged; and a clean close down attempted by transferring control to the shutdown routine with an identifying return code set. If a second program interrupt occurs, control is returned directly to CMS with a different return code set. The other processes call NCPINIT whenever they detect an irrecoverable error condition by deliberately causing a program interrupt, after logging the event.

The NCP virtual machine, after having been automatically established by CP, executes a 'profile' of CMS commands by means of the EXEC facility (Section 3). These commands cause the Network Control Program to be loaded from disk and to be entered at its start address. On control being returned to CMS, further commands in the profile cause a warning message to be sent to the CP console operator and either the re-loading of the NCP or the 'logging off' of its virtual machine according to the value of the return code and the number of times re-loading has been attempted.

4.5 The Treatment of Errors

It is considered that the major requirement of error recovery for the network system is to maintain effective connections, on behalf of users, for as long as possible and to notify them whenever a crash is imminent. A secondary requirement is to record those events which lead up to a crash, in order that remedial action can be taken in future versions of the system.

Three classes of error are detected; device errors associated with the IMP and VMCOM interfaces; system errors caused by faults in the program logic or the exhaustion of facilities; and protocol errors in the use of any of the software interfaces. A fourth class of error, outside the control of the virtual machine, is concerned with real machine failures. Detection and recovery procedures for this class are the

responsibility of CP.

IMP hardware errors, occurring during the initialization phase, result in an identifying message being typed on the NCP virtual machine console, a particular return code set and control returned to CMS. The profile, controlling the NCP virtual machine, attempts to re-initialize the system a number of times and if the failure persists notifies the CP operator so that the IMP can be investigated.

Device errors occurring after the initialization phase has been completed invariably cause a system crash. The type of error is then recorded in the NCP disk log and the crash routine in NCPINIT is entered. The crash routine attempts to issue an interrupt control message to all local users and a 'closing down' message to the IMP after sending a complete core dump to the (virtual) line printer. Any device errors detected by the routine are ignored.

System errors (Fig. 23) in any part of the Network Control Program also results in the event being logged and the crash routine in NCPINIT entered.

System errors are caused by such events as the failure to obtain a block of free store for new table or queue entries; the detection of an inconsistent entry or structure in one of the tables; the receipt of an interrupt from a device not in the PORT TABLE and the detection by CP of an attempt to perform an illegal operation.

Protocol errors in the use of the software interfaces (Fig. 24) never result directly in a system crash.

Most IMP-to-HOST messages are logged and any format errors from the IMP are ignored. Checks are carried out on the validity of all requests from foreign NCPs and any inconsistencies or errors in format are logged and an appropriate error message despatched to the originator. The validity of all requests by NET is checked and the status field in the reply control message set accordingly (Fig. 14). The NET routines tests the validity of requests from a process and replies with error codes as previously listed.

4.6 Concluding Remarks

This report has described the approach adopted by one site in implementing its network software. Other sites have adopted different approaches consistent with their own local requirements. The NCP developed by WHITE (Ref. 20) for the IBM 360/75 is accessed from a batch-mode job under the MVT operating system. Some sites directly modified their operating systems in order to incorporate the network facilities.

CODE	ADD INFO	REASON	CRASH
0	Header and 12 bytes	Code 11 (ERR) message detected from forNCP (NCPIMPI)	No
1	Header	Code 1 message detected from IMP (NCPIMPI)	No
2	Header	Code 3 message detected from IMP (NCPIMPI)	No
3	Header	Code 8 message detected from IMP (NCPIMPI)	No
4	Header	Code 9 message detected from IMP (NCPIMPI)	No
5	Header	We get a reset reply	No
6	------	SIO IMP IN INOPERABLE (NCPIMPI)	Yes
7	------	SIO IMP IN BUSY (NCPIMPI)	Yes
8	------	UE ON CC=1 IMP IN (NCPIMPI)	Yes
9	------	Other error CC=1 IMP IN (NCPIMPI)	Yes
10	------	Unexpected Bits IMP IN (NCPIMPI)	Yes
11	------	UE IMP IN (NCPIMPI)	Yes
12	------	Not DE on IMP IN (NCPIMPI)	Yes
13	------	Insufficient free store (NCPIMPI)	Yes
14	------	Our IMP dead (NCPIMPI)	soon
15	------	IMPS link table full (NCPIMPI)	No
16	------	NOP From NCP	No
17			
18			
19	Header and 16 bytes	We sent error message to NCP (NCPIMPI)	No
20	Header	We get a reset (NCPIMPI)	No
21	------	System error in CLS (NCPIMPI)	Yes
22	Header	New host sent to us	No
23	Header	FOR IMP/HOST DEAD (NCPIMPI)	No
24	Header	NOP from IMP	No
25		SIO IMP OUT Inoperable (NCPIMPO)	Yes
26	------	SIO IMP OUT Busy (NCPIMPO)	Yes
27	------	UE ON CC=1 IMPOUT (NCPIMPO)	Yes
28	------	Other error CC=1 IMPOUT (NCPIMPO)	Yes
29	------	Unexpected bits IMPOUT (NCPIMPO)	Yes
30	------	UE IMP OUT IMPOUT (NCPIMPO)	Yes
31	------	Interrupt Handler error NCPMAIN	Yes
32	XX/XX/XX	Date/time up	No
33	------	Card reader error (NCPXFER)	Yes
34	------	Insufficient free store (NCPXFER)	Yes
35	------	Software Table error (NCPXFER)	Yes

Fig. 23: NCP Error Codes

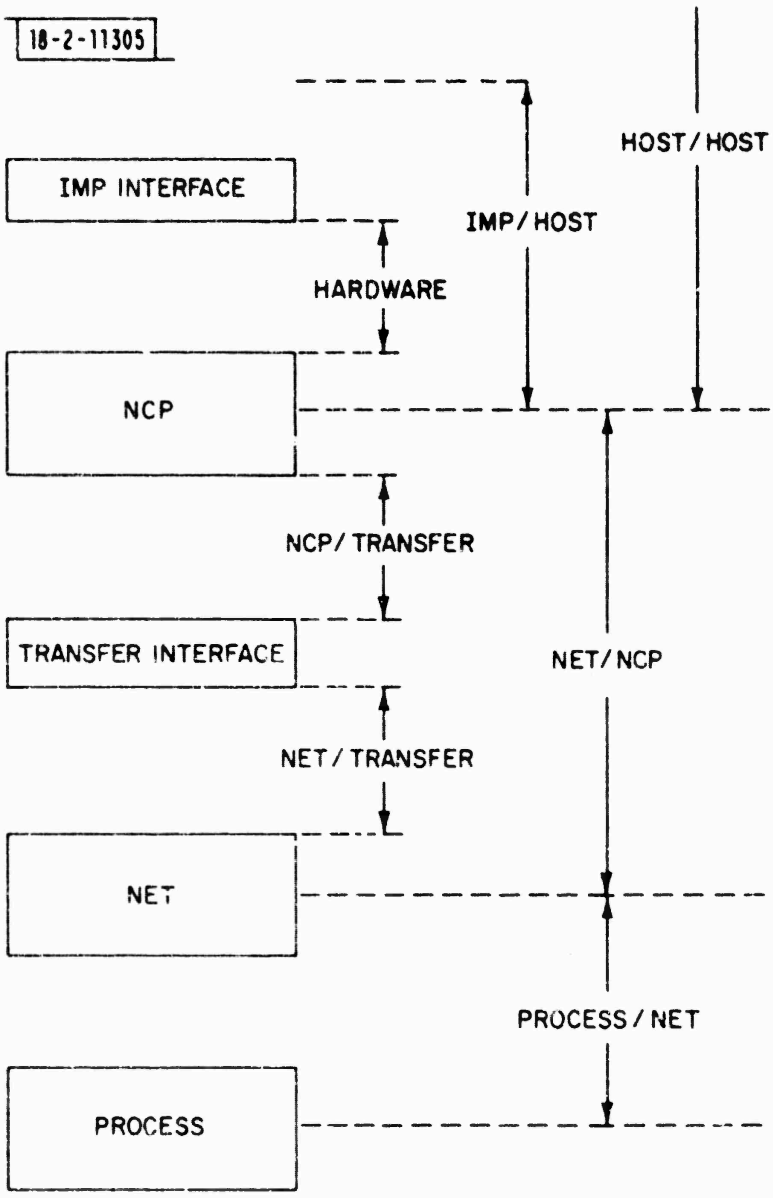


Fig. 24: Major Interfaces in the System

Subsequent to the original development of the network software, a number of changes have been made to the network. New HOSTs have been connected, bringing the total now to thirty-six, and a terminal IMP (TIP) has been introduced (ORNSTEIN, Ref. 13) to allow terminal users without a HOST to access the network. Some changes have been implemented in the HOST-to-IMP protocol (McKENZIE, Ref. 11) and proposals made for a different HOST-to-HOST protocol (WALDEN, Ref. 19). Further work has been carried out on the process-to-process protocols (CROCKER, Ref. 7), on the design of the network (McQUILLAN, Ref. 12), and on its performance measurement (COLE. Ref. 5). Future development and utilization has been discussed by ROBERTS (Ref. 16).

Some of these changes necessitated modifications to the NCP. In the case of new HOSTs, this simply requires the addition of new entries to the HOSTS WE KNOW table, but protocol changes require some modifications to the routines themselves. It is believed that the development of the NCP in a virtual machine and its modular structure has assisted the modification process.

Appendix A

IMP Interface

The IMP is connected to the 360 multiplexer channel via a duplex interface unit. The 360 READ command may be issued only to the input segment of the controller (address 60) and the 360 WRITE command may be issued only to the output segment of the controller (address 61).

The controller will not allow its Host Master Ready relay to close unless the CONTROL-ON command has been issued. This relay closure is tested by the IMP at all times. The 360 program must issue a CONTROL-ON to initiate IMP operation and must maintain such a command at all times. When terminating IMP operation the CONTROL-OFF command should be issued. This command renders the controller inert to all IMP service requests and no "Attention" interrupts will be sent to the 360.

The CONTROL-ON command the CONTROL-OFF command may be issued only to the read address. The NO-OP TEST I/O, and HALT I/O commands may be issued to either address. All other commands, or an improper combination, will then result in the command being rejected during initial selection. Unit Check (Bit 6) will appear in the initial status.

After the 360 program has issued a CONTROL-ON command the controller goes into a ready state. This condition will allow the input segment to receive data from the IMP. Should the IMP send a message to the input segment, the first 16 bits are shifted in. If no READ command had been issued by the 360 program an "Attention" interrupt is sent to the 360. The 360 program is expected to respond by issuing a READ command.

The status response in the CSW during initial selection may take the following format:

a. All zero for READ, WRITE, or TIO, indicating that the command was successfully initiated.
b. Unit Check, indicating an improper command. The command will have been rejected.
c. Unit Exception, indicating that the IMP Master Ready relay is not closed.
d. Device End/Channel End will be given in response to CONTROL-ON, CONTROL-OFF, and NO-OP. This is also considered as final status for these commands.
e. Status Modifier/Control Unit End/Busy will result if the controller is "Busy" at the time the command is issued.

Final Status will be given at the completion of all operations. It will consist of Device End/Channel End (DE/CE) for both READ and WRITE. Certain other conditions may be indicated, as follows:

a. Unit Exception will be given if the last data transfer in, or out, was interrupted by the IOP dropping its Master Ready signal, even momentarily. Unit Exception indicates that the data received, or sent, may be in error.
b. A Device End/Channel End will be given in response to Halt I/O if a command was indeed in operation at the time the Halt was issued.
c. Final status may include an Attention Bit along with DE/CE. This indicates that a subsequent READ must be issued to obtain data already within the input segment.

Appendix B

HOST-to-IMP Messages

A HOST-to-IMP message consists of not more than 8096 bits of information of which the first 32 bits are known as the 'leader'.

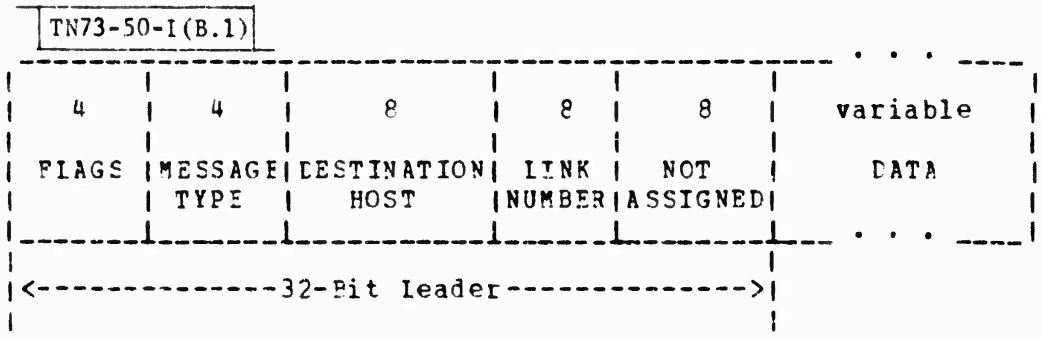


Fig. B.1: HOST-to-IMP Message Format

The leader is always present and defines the type of message, its destination and the link number, if any, with which it is to be associated. A description of each HOST-to-IMP message type follows:

a. Regular Message (Message Type 0).

Regular messages are forwarded through the network and presented to a destination HOST, specified by the DESTINATION HOST field in the leader, as a regular IMP-to-HOST message. Regular messages are the medium for all HOST-to-HOST communications, the DATA field being formatted according to the HOST-to-HOST conventions. Having issued a regular message on a particular link, that link is said to be 'blocked' and further messages cannot be sent until a RFNM or other 'unblocking' message has been received.

b. Error in Leader (Message Type 1).

If the NCP receives an IMP-to-HOST message with an invalid leader, it can respond with a Type 1 HOST-to-IMP message to advise the IMP of the format error.

c. HOST Going Down (Message Type 2).

The NCP should transmit a Type 2 HOST-to-IMP message about 10 seconds before voluntarily closing down. The IMP would then mark the HOST dead and advise the rest of the network accordingly. The 10-second delay is to allow time for the notification to reach all other IMPs in

the network and for messages already in transit to be processed.

d. No Operation (Message Type 4).

The IMP always discards this message which is intended for use during initialization. A number of Type 4 HOST-to-IMP messages are to be issued by the NCP, on re-activiation after a shutdown, to advise the IMP that the HOST is alive. The IMP will then relay this information to other IMPs in the network.

e. Regular Message for Discard (Message Type 5).

This message has all the properties of a Type 0 HOST-to-IMP regular message except that it is discarded by the destination IMP, before it reaches the destination HOST, and a Type 9 IMP-to-HOST message, rather than RFNM, is returned to the originating HOST to unblock the link. The message can be used for test purposes.

f. Error not in Leader (Message Type 8).

If the NCP receives an IMP-to-HOST message which has a valid leader but is otherwise in error with regard to the HOST-to-IMP protocol, it can respond with a Type 8 HOST-to-IMP message. The NCP might respond with such a message upon receiving an IMP-to-HOST message of excessive length.

g. All other HOST-to-IMP message types are undefined.

Appendix C

IMP-to-HOST Messages

A IMP-to-HOST message consists of not more than 8096 bits of information of which the first 32 bits are known as the 'leader'.

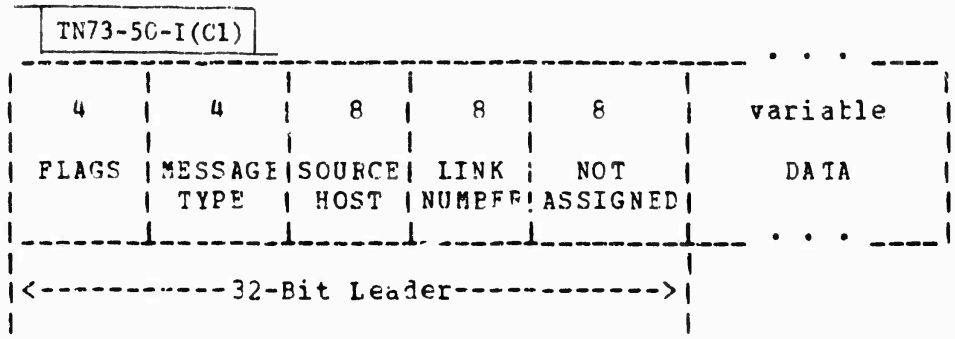


Fig. C.1: IMP-to-HOST Message Format

The leader is always present and defines the type of message, its source and link number, if any, with which it is associated. A description of each IMP-to-HOST message type follows:

a. Regular Message (Message Type 0).

   Type 0 IMP-to-HOST message, received by a destination HOST, originates as Type 0 HOST-to-IMP messages in the source HOST. All fields are preserved except that what originates as the destination HOST identifier in the HOST-to-IMP leader is converted to the source HOST identifier in the IMP-to-HOST leader.

b. Error in Leader (Message Type 1).

   The IMP issues a Type 1 IMP-to-HOST message on detecting an error in the leader of a previous HOST-to-IMP message. With the exception of the MESSAGE TYPE field, no significance is to be placed on the contents of the Type 1 leader.

c. IMP Going Down (Message Type 2).

   A Type 2 IMP-to-HOST message is issued by the IMP about 5 to 10 seconds before it voluntarily closes down. The delay is to allow notification of its impending shut down to propagate through the network and for messages already in transit to be processed. With the exception of the MESSAGE TYPE field, no significance is to be placed on the contents of the Type 2 leader.

d. Blocked Link (Message Type 3).

The IMP will issue a Type 3 IMP-to-HOST message whenever the NCP attempts to transmit a regular HOST-to-IMP message on a blocked link. With the exception of the MESSAGE TYPE field the contents of the leader will be identical to that of the discarded HOST-to-IMP message.

e. No Operation (Message Type 4).

The NCP is to discard all Type 4 messages from the IMP. These are intended for use during the initialization phase only.

f. Ready For Next Message (Message Type 5).

This message is used to notify the NCP that the first packet of a HOST-to-IMP regular message has been successfully passed to its destination HOST and that the link associated with it is now unblocked. The link is identified by the contents of the SOURCE HOST and LINK NUMBER fields in the leader of the Type 5 message.

g. Link Table Full (Message Type 6).

The IMP is capable of supporting at most 63 active transmit and 63 active receive links at any one time. A link is defined to be active if the NCP has issued a HOST-to-IMP regular message (or a Type 5 regular message for discard) over the link within a period of about 30 seconds. If the NCP attempts to exceed this limit the IMP will respond with a Type 6 message whose leader is identical, with the exception of the MESSAGE TYPE field, to that of the rejected HOST-to-IMP regular message.

h. Destination IMP or HOST Dead (Message Type 7).

This message is issued by the IMP if the NCP attempts to transmit a HOST-to-IMP regular message (or a Type 5 regular message for discard) to a HOST or IMP which is known to be dead. The leader of the Type 7 message is identical, with the exception of the MESSAGE TYPE field, to that of the rejected HOST-to-IMP regular message.

i. Error not in Leader (Message Type 8).

If the IMP receives a HOST-to-IMP message which has a valid leader but is otherwise in error with regard to the HOST-to-IMP protocol, it will respond with a Type 8 IMP-to-HOST message. This message has a leader which is identical, with the exception of the MESSAGE TYPE field, to that of the rejected HOST-to-IMP regular message. The IMP, for example, will issue a Type 8 message on receiving a HOST-to-IMP message of excessive length.

j. Incomplete Transmission (Message Type 9)

A Type 9 IMP-to-HOST message is issued to notify the NCP that the last HOST-to-IMP message sent over a particular link failed in transmission for some reason. The failure can be attributed to the unaccounted loss of some portion of the message; the loss of the associated RFNM; a message of excessive length or an excessive time required for re-assembly at its destination. A Type 9 IMP-to-HOST message is always returned in response a Type 5 HOST-to-IMP message, instead of a RFNM. Its receipt unblocks the link and its leader is identical, with the exception of the MESSAGE TYPE field, to that of the rejected HOST-to-IMP message.

k. All other IMP-to-HOST message types are undefined.

Appendix D

The HOST-to-HOST Protocol

All HOST-to-HOST communications are effected by regular HOST-to-IMP and IMP-to-HOST messages. Control messages are associated with link 0 and interprocess messages with non-zero links. The format of all HOST-to-HOST messages is as shown in Fig. 5.

Link Assignments:

Link numbers are assigned as follows:

a. Link 0 - Control Link.

b. Links 2-71 - Interprocess Communications.

c. Links 1, 72-190 - Reserved.

d. Link 191 - Network Measurement.

e. Links 192-255 - Private Experimental Use.

Control Messages:

Messages sent over the control link have the same format as other HOST-to-HOST messages. The connection byte size is to be 8 bits and the number of bytes is not to exceed 120. Control messages are always to consist of an integral number of control commands.

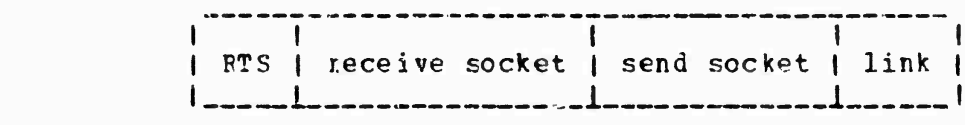
Control Commands:

Each control command begins with an 8-bit 'op-code', which defines the function of the command, followed by a set, in some cases null, of parameters which qualifies the function. A description of each control command follows:

a. NOP - No Operation.

The NOP Command can be issued at any time and is to be discarded by the receiving NCP. It is useful for formatting control messages and for determining, from the IMP-to-HOST reply, whether a foreign NCP is active or not.

b. RTS - Request for Connection, Receiver to Sender.



The RTS command is used to establish a connection and is sent from the HOST containing the receive socket to the HOST containing the send socket. The link number to be used by the interprocess connection is to be in the range 1-71 and is assigned by the issuing NCP. There is no prescribed lifetime for an RTS command which can be queued by the receiving NCP for an aribitrarily long period of time. The command could be cancelled by either NCP issuing a CLS (section C.3.d) and the connection is established when either the RTS matches a previously issued STR (section C.3.c) or when an STR is subsequently issued which matches the RTS.

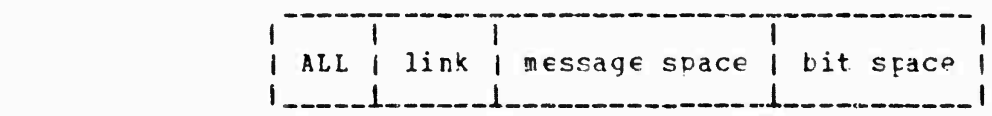
c. STR - Request for Connection, Sender to Receiver.

The STR command is used to establish a connection and is sent from the HOST containing the send socket to the HOST containing the receive socket. The connection byte size is to be in the range 1-255 and is assigned by the issuing NCP. There is no prescribed lifetime for an STR command which can be queued by the receiving NCP for an arbitrarily long period of time. The command can be cancelled by either NCP issuing a CLS (section C.3.d) and the connection is established when either the STR matches a previously issued RTS (section C.3.b) or when an RTS is subsequently issued which matches the STR.

d. CLS - Close.

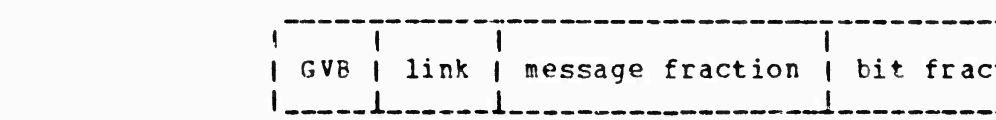
The CLS command is used to terminate a connection or to cancel a request for connection. The 'my socket' field contains the socket local to the issuing NCP and the 'your socket' field contains the socket local to the receiving NCP. There is no prescribed lifetime for a CLS command but it is required that an NCP reply with a matching CLS 'as quickly as possible' in order that the sockets can be released for further assignments

e. ALL - Allocate.



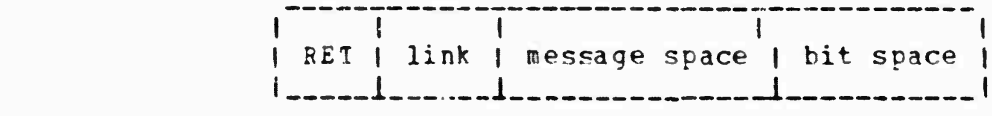
The ALL command is sent from a receiving NCP to a sending NCP to increase the allocation counters of the sending NCP. Allocate commands can be issued at any time while a connection is fully established. Interprocess messages can only be transmitted over a connection within the specified allocation.

f. GVB - Give Back.



This command is sent from a receiving NCP to a sending NCP to request that the sending NCP return all or part of its message and bit allocation for the connection. The fraction fields specify what portion, in 128ths, of each allocation is to be returned. The command can only be issued while a connection is fully established and the recipient NCP is to reply with a RET command (section C.3.g).

g. RET - Return Allocation.



This command is sent from a sending NCP to a receiving NCP either voluntarily or in response to a GVB command (section C.3.f). The command can only be issued while a connection is fully established.

h. INR - Interrupt by Receiver.

The INR command can be sent from a receiving NCP to a sending NCP when the receiving process wishes to interrupt the sending process. The command can only be issued while a connection is fully established.

i. INS - Interrupt by Sender.

The INS command can be sent from a sending NCP to a receiving NCP when the sending process wishes to interrupt the receiving process. The command can only be issued while a connection is fully established.

j. ECO - Echo Request.

The ECO command is used only for test purposes. An NCP could issue an ECO command at any time to another NCP and would expect to receive an ERP command (section C.3.k) in reply. The data field can be any bit pattern convenient to the sending NCP and the pattern is to be repeated in the data field of the ERP command.

k. ERP - Echo Reply.

The ERP command is to be issued by an NCP whenever it receives an echo request (section C.3.j). The data field of the ERP command is to be identical to the data field of the incoming ECP command.

l. EPR - Error Detected.

An NCP is permitted to issue an EPR command whenever it detectes a HOST-to-HOST protocol error by another NCP. The code field specifies the type of error and the data field provides additional information about the error. The following codes have been defined:

Code 0 - Undefined. An error for which no code exists has been detected. The data field is to be formatted according to the requirements of the originating NCP.

Code 1 - Illegal Op-code. An illegal op-code has been detected in a control message. The data field is to contain a copy of the command in error.

Code 2 - Short Parameter Space. The end of a control message has been encountered before all the required parameters have been found. The data field is to contain a copy of the command in error.

Code 3 - Bad Parameters. Incorrect parameters have been found in a control command. The data field is to contain a copy of the command in error.

Code 4 - Request on a Non-Existent Socket. A request, other than an RTS or STR, has been received associated with a link or socket for which no request for connection has been issued in either direction. The data field is to contain a copy of the command in error.

Code 5 - Socket or Link Not Connected. A request, other than an RTS or STR, has been received for a connection not fully established or for a link not yet assigned. The data field is to contain a copy of the command in error.

m. RST - Reset.

The RST command is used by one NCP to inform another that all information concerning existing connections between the two should be purged from the tables of the NCP receiving the RST. The sending NCP can expect to receive an RRP command (section C.3.n) in reply.

n. RRP - Reset Reply.

This command is to be issued by an NCP in reply to an RST command (section C.3.m).

o. All other op-codes are undefined.

Appendix E

CP Support for a Virtual Terminal Device

Modifications to CP have been made to support a 'software terminal', i.e., one driven by a virtual machine rather than by a real device such as a 2741. CP performs, via software, functions usually done by SIO's to a real terminal device. The purpose is to provide terminal-like access to CP for users wishing to log on to the system through the APPA network. The LOGGER virtual machine runs a program which interfaces between the network and CP virtual terminals.

Two new multiplexer devices (TYPNCPT and TYPNCPD) have been defined to support virtual terminals. An NCP terminal block (NCPT) has been added to the chain of real device blocks (RDEVBLKs) and a user logged on to such a terminal appears to CP like any other user except at the console (RDCONS and WRTCONS) level or lower. RDCONS and WRTCONS detect I/O to NCPTs and create CCW packages identical to those made up for 1052's.

Virtual terminal devices (NCPD's) are defined, through the directory, as part of the LOGGER virtual machine. I/O instructions from the LOGGER to an NCPD are interpreted according to a protocol for communicating with an NCPT. Initially, an addressed NCPD must be associated with an available NCPT so that a user may log on. Subsequently, information is transferred between the NCPD and an NCPT to satisfy CP read and write operations.

Initial connection is established by a HIO from the LOGGER virtual machine to one of its available NCP devices. This HIO causes CP to search for a free NCP terminal (in the chain of RDEVBLKs) and to establish a connection between the two. All subsequent I/O operations concern the transfer of data accross this connection until the link is broken.

A connection once established may be broken in one of two ways: by the user typing 'LOGO' or by the LOGGER virtual machine issuing a TIO to its NCP device. The second case is analogous to a user turning power off on a 2741 terminal, and may occur because the LOGGER crashed and came back up, or because a network link was closed. The first case is detected in the same way as a logoff from any other terminal device is detected. The LOGGER is informed of the broken connection by means of a unit check in his CSW.

A read-type command from the virtual machine to its NCP device is issued to pick up data from CP writes. The data in CP's write buffer is moved into the area beginning at the virtual machine's read address, and the read command is terminated with CE and DE. NCP support

follcws the protocol for a 1052 in that a 'locked keyboard' system is maintained. Thus, the virtual machine controlling NCP devices should always keep a read-type command active on these devices.

When CP puts up a read to an NCP terminal, the virtual machine is signalled to send data by a unit exception on the NCP device. This sequence is analogous to the write circle C/read circle D needed to unlock the keyboard of a 2741. On receipt of a unit exception, the LOGGER virtual machine should respond by issuing a write-type command to the NCP device.

The I/O operations defined for an NCP device, and the responses to them, are as follows.

TIO: used as a power off signal when a network connection has been closed.

CC=0 means a user was on and has been logged off.
CC=1 means no connection existed.

HIO: used as an attention signal. If no connection between the addressed NCP device and an NCP terminal exists, this routine will attempt to find an available NCP terminal and establish a connection. In this case,

CC=1 means a connection was established.
CC=2 means there is no available NCP terminal.

If a connection exists, HIO acts like an attention interrupt.

CC=0 means an ATTN has been simulated, and an interrupt is pending.
CC=1 means ATTN has been simulated, but the NCP device had no active command, so no interrupt will be received.

SIO: used to transmit and recieve data. Only read and write are defined.

CC=0 means OK, I/O started.
CC=1 means CSW stored. Meaningful bits are busy, attention (means warning is coming, and a read should be issued), unit check (means no link to an NCPT exists), unit exception (means CP wants to read, and user should write to his NCPD), and program check (means a read was issued when a write was required, or vice versa).
CC=2 means device busy.

Interrupts received after a successful SIO may be:

CE+DE: successful termination
CE+DE+UE: occurs only on a read-type command, and
CE+DE+UF+SM: means that CP is asking for a userid. It allows the LOGGER to monitor his user's logon synchronously with CP.
ATTN: occurs asynchronously when CP is waiting for the LOGGER to send data, i.e. when his keyboard is unlocked. It means a warning-type message wants to blast through.
CE+DE+UC: means user has logged off.

Appendix F

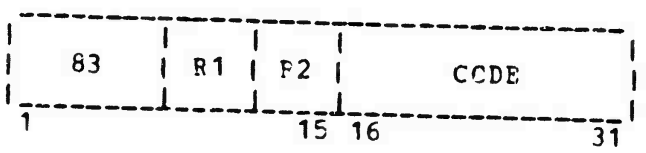
CP Virtual Machine Communications Facility

Purpose:

The VMCOM facility allows the virtual user to send data to another virtual user logged on to the CP system, or receive data transmitted from another virtual user.

Format:

The VMCOM diagnose instruction is defined as follows:



where:

R1 - Pointer to a Control Block
R2 - not used
CODE - (x'0050') used to designate the VMCOM request

Control Block:

Byte	Data
0	USERID (8)
8	Request flag (1)
9	Data buffer address (3)
12	Length of data buffer (2)
14	Number data bytes read (2)

Request Flag	Description
x'01'	Write data to user's stack (wait)
x'02'	Read data from own data stack
x'03'	Purge data from own data stack (all)
x'04'	Purge data sent to particular USERID
x'05'	Test for any data in own data stack
x'06'	Test for data sent to particular USERID
x'07'	Write data to user's stack (no wait)
x'80'	Lost message flag

Usage:

The receiver of a data write will receive a device end interrupt on virtual device x'fd' when some data has been placed on his data stack. The receiver should then initiate a read request to get the data into his virtual machine. For the writer of data, there is a limit of one data transfer between a virtual machine pair. If this limit is exceeded, a condition code 2 will be reflected on the data write attempt, and the receiver's control block will have the request flag byte set to b11'1xxxxxx' to indicate data lost. Data writes will write the whole data buffer out, data reads will read up to the buffer length and then indicate data buffer overflow. Null writes and null reads are allowed to communicate interrupts between virtual users. A write with no wait will be allowed for class 'C' users only.

Ending Conditions:

Condition Code	Description
CC0	Successful completion
CC1	Unsuccessful completion
	Control Block not full word aligned
	Control Block not within virtual core
	Data buffer not within virtual core
	Invalid request flag specified
	Data buffer > 4096 bytes
	Not a class 'C' user, request invalid
CC2	Data buffer overflow on read request
	Unprocessed data in receiver's data stack
CC3	User not on system
	Unable to reflect interrupt to receiver
	No data in stack on read request

Appendix G

The NET Routine

Purpose:

To communicate with the NCP to issue NCP control commands or to receive a message which has been buffered by the NCP

Calling Sequence:

The user makes a subroutine call of the form:

```
LA      1,NETPLIST
L       15,=A(NET)
BALR    14,15
LTR     15,15
BNZ     15,ERROR
```

where NETPLIST is of the form:

```
CL4     'Control Code'
XL3     'My ID'
XL1     'My Tag'
XL1     'For Host'
XL3     'For ID'
XL1     'For Tag'
XL1     'Flag'
H       'Bit count'
AL4     (Buffer Address)
F       'Status Reply'
F       'Bits Left'
H       'Messages left'
X       'Byte Size'
X       'Spare'
AL4     (Address of interrupt stack)
```

and where Control Codes are:

```
CON     Connect
LIS     Listen
SND     Send
RCV     Receive
CLS     Close
STA     Request Status of Socket
INT     Sent interrupt
NOP     returns net status.  Useful after wait.
ENC     Enable connect
ENL     Enable listen
DIS     Disable
PUR     Purge
```

Usage:

A send buffer must contain an 80-byte header followed by up to 1000 bytes of data. The NET routine will use the 80-byte header to build an NCP control message. Similarly, a receive buffer must be 1080 bytes long. The first 80 bytes will be used for the NCP control message and the rest of the buffer will be used to store the text of any message which was held by the NCP.

Before any call to NET is made, the interrupt stack pointer should be cleared. On return from a call to NET the interrupt stack pointer should be checked. If it is zero, no interrupts were received. A non-zero pointer indicates that an interrupt was received and the interrupt stack control block should be analyzed and then returned to free storage. The Interrupt Stack Control Block is shown in Fig. G.1 and the Interrupt Codes are listed in Fig. G.2.

Errors:

Control is always returned from the NET package with a code in general purpose register 15. The return codes are as follows:

0 Command accepted by NCP. Tag status given in reply.
1 Command rejected by NCP. Tag status given in reply indicates reason for rejection.
2 NETPLIST control code invalid.
Status reply not significant.
3 VMCOM failure.
4 Nothing there (NCP Special Case).
5 Unexpected message received by NET.
6 NCP error.
Reply does not match socket number issued.
8 VMCOM read error.
9 VMCOM send error.
10 No more free store for NET.
11 NCP just died.
12 Zero bit count.

TN73-50-I(G.1)

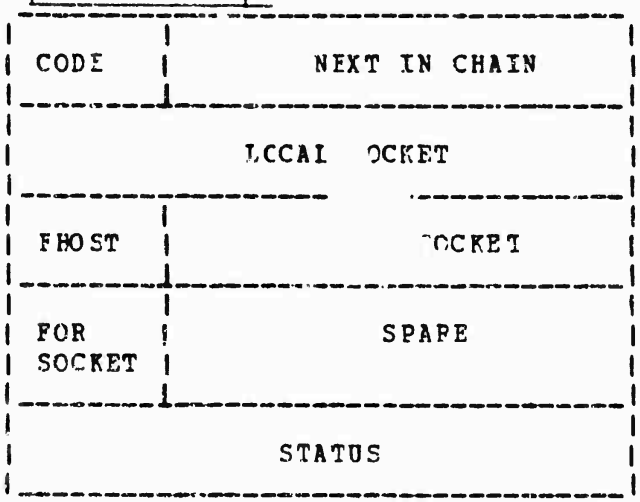


Fig. G.1: Interrupt Stack Control Block

TN73-50-I(G.2)

CODE	MEANING
0	Interrupt received. Could be SND or RCV socket. Status must be issued to clear this bit.
1	His listen issued on our SND socket. After the connection has been completed.
2	Receive buffer now loaded <u>after</u> issuing our listen (His connect implied).
3	His CLS received before we have issued ours.
4	Enabled connect matched by his connect.
5	Enabled listen matched by his connect.
6	NCP Died.
7	Foreign NCP Reset (RST received).

Fig. G.2: Interrupt Codes

Appendix H

NET User Macros

General:

A link block must be defined for each link required. This block contains local and foreign socket numbers together with status and flag areas. The NETPLIST macro is a DSECT which specifies the format. Link blocks are defined by NETSOCK or NETSOCKV macros, and refer either to send or receive links. The user's network id must be obtained with the NETID macro which obtains a user number and places it in the appropriate portion of the socket number.

The link must then be activated via a NETOPEN macro.

Transfers are achieved via NETREAD and NETWRITE macros and the link may be closed via the NETCLOSE macro.

Link status may be obtained using NETSTAT and network interrupts may be sent using NETINT. NETNOP obtains the current flag state without accessing the NCP.

Macros:

NETSOCK (&A,&B,&C)

Defines a link block. It requires a label which is used by all other macros to identify the link. It has three parameters:

&A - a 1-byte hex value of the local tag
&B - a 1-byte hex value of the foreign host
&C - a 4-byte hex value of the foreign socket

NETSOCKV (&A,&B,&C)

Is an alternative to NETSOCK. It also defines a link block and requires a label for link identification purposes. It has three parameters.

&A - a label for the 1-byte local tag
&B - a label for the 1-byte foreign host
&C - a label for the 4-byte foreign socket

Before using the link specified by NETSOCKV values must be moved into the locations named by &A, &B and &C. For example:

```
          MVI TAG,X'01'
          MVI FHOST,X'0A'
          MVC FSOC(4),=X'00000001'
           .
           .
           .
LINK1     NETSOCKV TAG,FHOST,FSOC
```

The following macros require the link block label as the first parameter (&A):

NETID — Obtains user's network id and places it in the link block

NETOPEN — Assigns a link and establishes the connection.

NETOPENE — Used after a NETENA to open a connection

NETREAD — Transfers into the specified buffers any outstanding data in the link (max 1000 bytes)

NETWRITE — Transfers from specified buffers to network. (max 1000 bytes)

NETSTAT — Obtains network status of link

NET. — Issues appropriate network interrupt

NETINTP — Obtains current interrupt state

NETCLOSE — Closes the connection

NETENA — Defines a local socket for which a foreign host can initiate a connection

NETDISA — Releases a local socket which was enabled

NETPURGE — Causes the entry in the NCP RV table to be deleted for a closed connection which has not been closed by the foreign host

The NETREAD and NETWRITE macros have two other parameters. &B is the address of a full word containing the buffer address. &C is the address of a full word containing the bit count.

A NETWAIT may be issued to wait for an NCP interrupt. When the wait completes, a NET NOP may be issued and the current interrupt(s) will be placed in the interrupt stack.

Return Codes:

Return codes in GPR15 are:

0 - Operation completed satisfactorily - Status field significant

1 - Operation denied by network. Status field significant

2 - User made logical error. Status not significant

3 - System Failure. Status not significant

Appendix I

The MLCTST Program

Purpose:

This program is used to issue CCW's to the MLC/IMP. The program has a one-character command language. It is useful to issue these commands on one line separated by the logical break character (#).

Requests:

I (INPUT) - To set the current I/O device to the IMP's input device

O (OUTPUT) - To set the current I/O device to the IMP's output device.

N (ON) - To turn the IMP on. This request must be issued on the IMP's input device.

F (OFF) - To turn the IMP off. This request must also be issued on the IMP's input device.

R (READ) - To issue a read SIO. If a read is already outstanding, a CC=2 (busy) results. When the read completes, the data read will be in an input buffer which can be displayed or dumped. A read on the output device causes a CC=1 with the CSW stored indicating a program check.

W (WRITE) - To issue a write SIO. The data written is set up to be a IMP NOP (a 4-byte leader only). A write on the input device causes a CC=1 with the CSW stored indicating a program check.

H (HIO) - To issue a HIO instruction to the current device.

T (TIO) - To issue a TIO instruction to the current device

X (NOP) - To issue a NOP CCW to the current device.

V (invalid) - To issue an invalid CCW operation code to the current device.

D (DISPLAY) - To dump the input buffer which contains the data last read. CP is used to display the storage area.

P (PAUSE) - To issue a CMS WAIT on the input device to wait for an interrupt.

Q (QUIT) - To leave the MLCTST program.

Appendix J

The TRYNET Program

Purpose:

To test an NCP using the NET routine.

Requests:

```
CON  ls fh fs bs
LIS  ls fh fs bs
SND  ls message
RCV  ls
INT  ls
CLS  ls
STA  ls
ENC  ls
ENL  ls
DIS  ls
PUR  ls
END
```

where:

ls - is a local socket (8 hex digits)

fh - is a foreign host (2 hex digits)

fs - is a foreign socket (8 hex digits)

bs - is the byte size for the connection (2 hex digits)

message - is a string of characters to be sent over the connection

Usage:

Requests are issued from the terminal tc perform a basic NCP function. For each request issued, the RV table entry is returned if one exists. The second two status bytes are printed indicating the status of the socket connection.

Whenever an interrupt is received, a line is printed giving the type of interrupt. When an RCV is issued, the message received, if any, is also printed. If an NCP communications message is lost (VMCOM interrupt lost), this fact is also printed. The user should then issue a status on each of the active sockets to determine the interrupt which was lost.

Appendix K

The TELNET LOGGER/SERVER

ICP Connection:

The TELNET LOGGER/SERVER follows the ICP protocol for making a pair of connections. The LOGGER is initially enabled for a connection on socket X'00000001'. When an RFC is received for this socket a pair of sockets will be chosen for the TELNET connections. If the maximum number of TELNET users which can be served are active, the initial connection is refused. Currently, three TELNET users can be served.

TELNET LOGGER:

After the ICP connections have been setup, the LOGGER expects a TELNET data type code, a string of network ASCII characters, or a null line (just CR-LF) to indicate whether its operation should be in ASCII or in EBCDIC character codes. ASCII is assumed unless the first byte received is the TELNET EBCDIC data type code (X'A2). When something has been received, the message:

Lincoln Laboratory CP/CMS Online

will be transmitted by the LOGGER. For example, if ASCII operation is desired a null line (just CR-LF) transmitted on the send socket will cause the welcoming message to be sent in ASCII. The CP login procedure can then begin. If communications is desired to be carried on with EBCDIC character codes, the first byte transmitted should be the TELNET data type code for EBCDIC (X'A2'). Thereafter all communications will be in the code originally used.

The CP login procedure expects the user to enter:

LOGIN userid

where the userid specifies the desired virtual machine. CP then replies with:

ENTER PASSWORD:

followed by the EBCDIC code for bypass (X'24') which is mapped into the TELNET code hide-your-input.

The user should then enter a password. Passwords entered from the network may be different from those entered from a local terminal. The LOGGER maps network passwords into a corresponding CP password. Thus, access to an account can only be made from the network if a network password, together with a CP password and userid, is entered into a file which is read by the LOGGER. If a userid entered from the network is not in

the LOGGER FILE (or if the network password does not match the one included in the file for the specified userid) the LOGGER passes an invalid userid (or password) to CP. The CP response for an invalid userid or password is then sent to the network user.

After a password is received by CP, CP transmits the EBCDIC code for restore (X'14') which is mapped into the TELNET control noecho.

TELNET SERVER:

Since the CP/CMS system operates with EBCDIC codes, ASCII codes must be translated into EBCDIC before being sent to a virtual machine. Figure K.1 gives the ASCII codes and their EBCDIC mapping. When the ASCII sequence CR-LF is received, it is mapped into the EBCDIC code NL. Whenever the TELNET control NOP is included in an input string, it is mapped into an EBCDIC idle (X'17') and then removed from the string. Thus, if TELNET NOP codes are included between a CR and LF, they are removed before the CR-LF is mapped into the EBCDIC NL.

The TELNET control hide-your-input is mapped into the EBCDIC code for bypass (X'24') and the TELNET control echo is mapped into the EBCDIC control for restore (X'14'). If the TELNET control echo is received, the SERVER should send the control noecho but this feature has not yet been implemented. Instead, the TELNET control echo is mapped into the EBCDIC code X'23'. If the TELNET break is received, it is interpreted as an attention signal and the appropriate action is taken by CP or CMS.

CP/CMS is a line at a time system and expects all input to consist of lines ending with a NL code. Characters received are buffered until the newline code is received.

Since CP/CMS is also a half duplex system, characters are only examined when the system is expecting input. If the system is not expecting input, a network interrupt is required to cause the SERVER to process received characters. On receipt of a network interrupt, characters received before the TELNET data mark is received are examined and discarded, except that if a TELNET break code is found, the appropriate CP/CMS interrupt action is stimulated.

On output, EBCDIC codes are mapped into network ASCII if a mapping exists; otherwise, the codes are mapped into the TELNET control NOP. A NL code is mapped into CR-LF. The EBCDIC code for bypass maps into the TELNET control hide-your-input and the EBCDIC code for restore maps into the TELNET control noecho. Also, the code X'23' maps into the TELNET control echo and the code X'38' maps into the TELNET control break.

Since CP/CMS is a line at a time, half duplex system the TELNET control break is transmitted as an end of message signal and also as an input prompt code. If characters were output without a NL, the break, as an end of message code, indicates to the user TELNET operating on a line at a time mode that the characters previously transmitted should be printed without waiting for the end of line sequence. If the user TELNET is also operating in a half duplex mode, the break as an input prompt indicates that the system is ready for input.

If input had been anticipated and sent by a full duplex user TELNET, the TELNET SERVER will have that input available for immediate processing. Thus, in the case of a full duplex user TELNET the break as a prompt should be ignored.

Though CP/CMS operates in a half duplex mode, it supports half duplex terminals with the reverse break feature allowing the system to abort an input mode in order to transmit a priority output message. In this situation, the TELNET SERVER transmits a TELNET SYNC. A half duplex user TELNET should interpret this by aborting the input mode, i.e., revoking a previous TELNET break which was interpreted as an input prompt.

No codes in the output character stream can cause the TELNET data mark to be transmitted.

LOGOUT:

When a user logs out from his virtual machine, CP passes the equivalent of a line disconnect to the LOGGER. The LOGGER then closes the TELNET send and receive sockets.

TN73-50-I(K.1)

ASCII DEC	ASCII OCT	ASCII HEX	SYMBOLS	EBCDIC HEX	EBCDIC DEC
0	0	(00)	NUL	(00)	00
1	1	(01)	SOH	(01)	01
2	2	(02)	STX	(02)	02
3	3	(03)	ETX	(03)	03
4	4	(04)	EOT	(37)	55
5	5	(05)	ENQ	(2D)	45
6	6	(06)	ACK	(2E)	46
7	7	(07)	BEL	(2F)	47
8	10	(08)	BS	(16)	22
9	11	(09)	HT	(05)	05
10	12	(0A)	LF	(25)	37
11	13	(0B)	VT	(0B)	11
12	14	(0C)	FF	(0C)	12
13	15	(0D)	CR	(0D)	13
14	16	(0E)	SO	(0E)	14
15	17	(0F)	SI	(0F)	15
16	20	(10)	DLE	(10)	16
17	21	(11)	DC1	(11)	17
18	22	(12)	DC2	(12)	18
19	23	(13)	DC3	(13)	19
20	24	(14)	DC4	(3C)	60
21	25	(15)	NAK	(3D)	61
22	26	(16)	SYN	(32)	50
23	27	(17)	ETB	(26)	38
24	30	(18)	CAN	(18)	24
25	31	(19)	EM	(19)	25
26	32	(1A)	SUB	(3F)	63
27	33	(1B)	CTL	(27)	39
28	34	(1C)	FS	(1C)	28
29	35	(1D)	GS	(1D)	29
30	36	(1E)	RS	(1E)	30
31	37	(1F)	US	(1F)	31

Fig. K.1: LOGGER ASCII, EBCDIC Code Mapping

ASCII DEC	ASCII OCT	ASCII HEX	SYMBOLS	EBCDIC HEX	EBCDIC DEC
32	40	(20)	SP	(4C)	64
33	41	(21)	!	(5A)	90
34	42	(22)	"	(7F)	127
35	43	(23)	#	(7E)	123
36	44	(24)	$	(5B)	91
37	45	(25)	%	(6C)	108
38	46	(26)	&	(50)	80
39	47	(27)	'	(7D)	124
40	50	(28)	(	(4D)	77
41	51	(29)	)	(5D)	93
42	52	(2A)	*	(5C)	92
43	53	(2B)	+	(4E)	78
44	54	(2C)	,	(6D)	109
45	55	(2D)	-	(	96
46	56	(2E)	.	(4)	75
47	57	(2F)	/	(61)	97
48	60	(30)	0	(F0)	240
49	61	(31)	1	(F1)	241
50	62	(32)	2	(F2)	242
51	63	(33)	3	(F3)	243
52	64	(34)	4	(F4)	244
53	65	(35)	5	(F5)	245
54	66	(36)	6	(F6)	246
55	67	(37)	7	(F7)	247
56	70	(38)	8	(F8)	248
57	71	(39)	9	(F9)	249
58	72	(3A)	:	(7A)	122
59	73	(3B)	;	(5E)	94
60	74	(3C)	<	(4C)	76
61	75	(3D)	=	(7E)	126
62	76	(3E)	>	(6E)	110
63	77	(3F)	?	(6F)	111

Fig. K.1: LOGGER ASCII/EBCDIC Code Mapping (Continued)

ASCII DEC	ASCII OCT	ASCII HEX	SYMBOLS	EBCDIC HEX	EBCDIC DEC	
64	100	(40)	@	(7C)	124	
65	101	(41)	A	(C1)	193	
66	102	(42)	B	(C2)	194	
67	103	(43)	C	(C3)	195	
68	104	(44)	D	(C4)	196	
69	105	(45)	E	(C5)	197	
70	106	(46)	F	(C6)	198	
71	107	(47)	G	(C7)	199	
72	110	(48)	H	(C8)	200	
73	111	(49)	I	(C9)	201	
74	112	(4A)	J	(D1)	209	
75	113	(4B)	K	(D2)	210	
76	114	(4C)	L	(D3)	211	
77	115	(4D)	M	(D4)	212	
78	116	(4E)	N	(D5)	213	
79	117	(4F)	O	(D6)	214	
80	120	(50)	P	(D7)	215	
81	121	(51)	Q	(D8)	216	
82	122	(52)	R	(D9)	217	
83	123	(53)	S	(E2)	226	
84	124	(54)	T	(E3)	227	
85	125	(55)	U	(E4)	228	
86	126	(56)	V	(E5)	229	
87	127	(57)	W	(E6)	230	
88	130	(58)	X	(E7)	231	
89	131	(59)	Y	(E8)	232	
90	132	(5A)	Z	(E9)	233	
91	133	(5B)	[	(AD)	173	
92	134	(5C)	\	(4A)	74	(BACK-SLASH)
93	135	(5D)	]	(BD)	189	
94	136	(5E)	^	(71)	113	(CARAT)
95	137	(5F)	_	(6D)	109	

Fig. K.1: LOGGER ASCII/EBCDIC Code Mapping (Continued)

TN73-50-I(K.1)

ASCII DEC	ASCII OCT	ASCII HEX	SYMBOLS	EBCDIC HEX	EBCDIC DEC	
96	140	(60)		(79)	121	(GRAVE)
97	141	(61)	a	(81)	129	
98	142	(62)	b	(82)	130	
99	143	(63)	c	(83)	131	
100	144	(64)	d	(84)	132	
101	145	(65)	e	(85)	133	
102	146	(66)	f	(86)	134	
103	147	(67)	g	(87)	135	
104	150	(68)	h	(88)	136	
105	151	(69)	i	(89)	137	
106	152	(6A)	j	(91)	145	
107	153	(6B)	k	(92)	146	
108	154	(6C)	l	(93)	147	
109	155	(6D)	m	(94)	148	
110	156	(6E)	n	(95)	149	
111	157	(6F)	o	(96)	150	
112	160	(70)	p	(97)	151	
113	161	(71)	q	(98)	152	
114	162	(72)	r	(99)	153	
115	163	(73)	s	(A2)	162	
116	164	(74)	t	(A3)	163	
117	165	(75)	u	(A4)	164	
118	166	(76)	v	(A5)	165	
119	167	(77)	w	(A6)	166	
120	170	(78)	x	(A7)	167	
121	171	(79)	y	(A8)	168	
122	172	(7A)	z	(A9)	169	
123	173	(7B)	{	(8B)	139	
124	174	(7C)	|	(4F)	79	(BAR/OR)
125	175	(7D)	}	(9B)	155	
126	176	(7E)	¬	(5F)	95	(TILDE/NOT)
127	177	(7F)	DEL	(07	7	

ASCII DEC	ASCII OCT	ASCII HEX	TELNET CONTROLS	EBCDIC HEX	EBCDIC DEC	
128	100	(80)	DATA-MARK	(80)	128	
129	101	(81)	BREAK	(38)	56	
130	102	(82)	NOP	(17)	23	IDLE
131	103	(83)	NOECHO	(14)	20	RESTORE
132	104	(84)	ECHO	(23)	35	
133	105	(85)	HIDE-YOUR-INPUT	(24)	36	BYPASS

Fig. K.1: LOGGER ASCII/EBCDIC Code Mapping
(Continued)

Appendix L

The User TELNET

Purpose:

To access another terminal-oriented system on the ARPA network.

Format:

TELNET host <tag> RESUME EBCDIC HALFDUP
1 OPEN ASCII FULLDUP

host - either the hexadecimal code for a foreign network service site or a standard mnemonic for a foreign site. See Figure 1.

tag - the identifier for the local connections to the network. The tag is used together with the address of the virtual machine descriptor table (UTABLE) to form local socket numbers which are used in the network protocol.

RESUME - used to reactivate communications with a foreign site after having previously left the TELNET command leaving the connections open.

EBCDIC - to communicate with EBCDIC codes. The default is network ASCII.

HALFDUP - to operate under a half duplex protocol, i.e. with a locked keyboard.

The EBCDIC HALFDUP the protocol assumes that the TELNET break code (circle C) will be received to indicate when the keyboard should be locked for input.

In ASCII HALFDUP the keyboard will lock after a line of input and will unlock after one or more lines have been received for output. An external interrupt will also unlock a locked keyboard.

The default is full duplex where the keyboard is alwasy unlocked for input. A null line is required to temporarily lock the keyboard in order to receive output.

Usage:

A number of hosts on the ARPA network provide TELNET service. A Network Virtual Terminal (NVT) has been specified so that using sites can write one TELNET program which maps a local terminal into the NVT to access any serving site on the network. Once communication has been established between a using site and a serving site, keyed input is sent to the serving system and output from the serving site, when received, is typed on the local terminal.

The NVT protocol requires that the keyboard be capable of entering all of the 128 ASCII codes together with a number of other TELNET control codes. To support an NVT with an IBM 2741 terminal, it is necessary to adapt a control convention for entering codes which are not associated with single keys on the keyboard. In addition, since CP/CMS processes input from a 2741 on a line at a time terminated with a newline, a means must be established for entering a sequence of characters for transmission which is not terminated with a newline code.

When TELNET is initiated the message

ENTER CONTROL CHARACTER

is typed. A non-blank character should then be entered which defines the character which, in combination with another character, will be used to enter codes not associated with single keys. The control character is also used for other special control functions as described below.

Codes:

The NVT usually requires that characters be transmitted in an eight-bit ASCII code. Since the TELNET command is written to process EBCDIC codes ASCII codes received are translated into EBCDIC and characters to be transmitted are translated into ASCII before being sent to a serving site. Figure 2 gives the complete definition of EBCDIC indicating the EBCDIC controls and EBCDIC graphics. Figure 3 gives the codes for the ASCII controls and graphics. The complete mapping between 8-bit EBCDIC codes and 8-bit network ASCII codes is shown in Figure 4. The EBCDIC newline code (NL) is mapped into the ASCII codes for the pair of characters CR-LF.

The following ASCII/EBCDIC mapping is used for the non-EBCDIC graphics:

	ASCII		EBCDIC	
TILDE	(7E)	=	(A1)	NOT
BAR	(7C)	=	(6A)	OR
BACK SLASH	(5C)	=	(E0)	
CARAT	(5F)	=	(71)	
GRAVE	(60)	=	(79)	
LEFT BRACE	(7B)	=	(8B)	
RIGHT BRACE	(7D)	=	(9B)	
LEFT BRACKET	(5B)	=	(AD)	
RIGHT BRACKET	(5D)	=	(BD)	

The ASCII control DC3 (X'13') maps to the EBCDIC control TM (X'13'). The ASCII control NUL (X'00') is sent to the terminal as the EBCDIC code for NULL (X'00') and is not mapped into an IDLE (X'17').

The TELNET control hide-your-input is mapped into the EBCDIC code for bypass (print supress) and the TELNET control noecho is mapped into the EBCDIC code for restore (print restore). If the TELNET control for echo is received, a message is printed and it is mapped into an IDLE. Similiarly, if the TELNET control for break is received, a message is printed and it is mapped into an IDLE unless operation is in EBCDIC HALFDUP mode in which case the break is used to indicate that any received characters should be printed and the keyboard unlocked for input. If a data mark or an interrupt is received, no action is taken except to print a message to notify the user of this occurance.

Input:

When the control character is entered, the following character is mapped into a different code than that which it is normally mapped into, except when the following character is a space or a character not defined to have a meaning when preceded by the control character. Figure 5 gives the mapping of the characters on a 2741 keyboard when preceded by a control character. The following 2741 keyboard characters do not have a different meaning when preceded by the control character.

```
$ # * % @
+ - = _
. , : ;
! | ? ¢
SPACE
BACKSPACE
TAB
```

When a character is mapped into its control code, the control character is mapped into the code for IDLE. If the control character is entered as the last character before the newline key is entered, the sequence of characters entered is transmitted without the newline code. That is, the newline code is not transmitted when it is preceded by the control character.

When the 2741 keyboard is unlocked for input, characters received cannot be typed until the keyboard is locked again. After a line is entered, received characters can then be typed. When operating in full duplex or ASCII half duplex, a null line entered will allow received characters to be typed but will not cause the new line code to be transmitted. To cause a null line, i.e., just the new line code to be transmitted, the control character should be entered as the only character in the input line. In EBCDIC HALFDUP a null line entered will cause a null line to be transmitted.

Output:

ASCII output received from the NVT is converted into EBCDIC with the sequences CR-LF converted into IDLE-NL. The EBCDIC characters are then sent to the terminal. Note that not all 128 ASCII codes when converted to EBCDIC will print on a 2741. Of the 95 ASCII graphics and the 8 ASCII controls which are defined for the NVT printer, the following are not visible or audible:

CARAT
GRAVE
BACK SLASH
LEFT BRACE
RIGHT BRACE
LEFT BRACKET
RIGHT BRACKET
ASCII CONTROL BELL (BEL)
ASCII CONTROL VERTICAL TAB (HT)
ASCII CONTROL FORM FEED (FF)
ASCII CONTROL CARRIAGE RETURN (CR)

Figure 6 shows how the EBCDIC codes from X'40' through X'FF' will appear on a 2741 terminal. Figure 7 shows how the EBCDIC codes will appear when printed with a PN train on the offline printer and Figure 8 shows how these codes appear when printed with a TN train.

Controls:

If the first character in an input line is the control character and the next character is a space, the rest of the line is interpreted as a TELNET control command. A control command consists of a control word and parameters separated by spaces. Controls are defined which permit TELNET controls to be transmitted to the serving site, allow input to come from a file or output to go to a file, allow CMS functions or transient commands to be issued, redefine the control character or TELNET mode, close connections or leave the TELNET command with the connections still open, as well as controls to support a reader, punch, and printer with RJS operation. The controls are described below.

CONTROL x

Where x is the new control character

CLOSE

To close all connections and quit

QUIT

To leave TELNET

EBCDIC

To go into transparent mode, i.e., no translation

ASCII

To translate input and output to network ASCII

BREAK

To send the TELNET break code

SYNC

To send the TELNET data mark code and an interrupt

AIT:

To send a TELNET break and a SYNC

HIDE-YOUR-INPUT

To send the TELNET hide you input code.

NOECHO

To send the TELNET noecho code

ECHO

To send the TELNET echo code.

CMS command arg1 . . . argN

To issue CMS core resident function or transient command.

INPUT fn ft
* TERMIN
* *

To get input from a file If fn is defaulted, input is reset to come from the terminal. If fn is * file input resumes after the last line read. After an EOF, the next line read will be the first line of the file.

An external interrupt while input is coming from a file will cause the line number of the next line to be read from the file to be typed and input to be reset to come from the terminal.

```
OUTPUT fn OFF  TERM   INPUT   INCUT
       *  ON   NOTERM NOINPUT OUTPUT
```

To write output to the file 'fn TERMOUT'. If fn is defaulted, output is reset to go to the terminal. If fn is *, file OUTPUT is resumed with the same options as were last used.

For Output to the Terminal:

If the last character is a CR, a line with just the control character is typed on the next line (with a NL)

If the last character is not a NL or a CR, the line is typed without a NL (i.e., with TYPE).

For Output to a File:

If just a NL is in the line, just the control character is sent to the file.

If the last CHAR is not NL or CR, the control character is added after the last character, except if 130 characters must be sent to the file.

If the last CHAR is a CR, it is included in the file.

OFF causes all output to be discarded.

ON is the default, and causes output to go to the terminal.

TERM causes output to also go to the terminal.

NOTERM is the default, and causes output to go the the file but not to the terminal.

OUTPUT is the default and causes just terminal output to be put to the file 'FN termout'.

INOUT causes both terminal input and terminal output to be put to the output file.

INPUT causes terminal input but not output to be put to the output file.

NOINPUT is defaulted and causes input to not go to the file.

PURGE

To purge all output currently received by the NCP. This control is not currently implemented.

READER fn ft
* READER

To send a job to the RJS system at UCLA's CCN.

If fn and ft are defaulted, input will come from the card reader.

PRINTER fn ft
* PRINTER

To receive printer output from the RJS system at UCLA's CCN.

To receive punch output from the RJS system at UCLA's CCN.

If fn and ft are defaulted, output goes to the printer.

PUNCH fn ft
* PUNCH

If fn and ft are defaulted, output goes to the punch.

HOST	SITE	MACHINE	SYSTEM	HOST NUMBER DEC	OCT	HEX
NMC	UCLA	SIGNA-7	SEX	1	1	01
ARC	SRI	PDP-10	NIC	2	2	02
UCSB	UCSB	360/75	OS/MVT	3	3	03
UTAH	UTAH	PDP-10	TENEX	4	4	04
MULTICS	MIT	H-645	MULTICS	6	6	06
SDC	SDC	370/155	ADEPT	8	10	08
HARV	HARVARD	PDP-10	4S7x	9	11	09
LL	LL	360/67	CP/CMS	10	12	0A
CASE	CASE	PDP-10	10/50	13	15	0D
CMU	CMU	PDP-10	TOPS-10	14	16	0E
ILLIAC	AMES	B-6500	?	15	17	0F
AMES	AMES	360/67	TSS/360	16	18	10
CCN	UCLA	360/91	OS/MVT	65	101	41
SRI	SRI-AI	PDP-10	TENEX	66	102	42
BBNA	BBN	PDP-10	TENEX	69	105	45
DMCG	MIT	PDP-10	ITS	70	106	46
RAND	RAND-FCC	PDP-10	TENEX	71	107	47
TX2	LL	TX-2	APEX	74	112	4A
BBNE	BBN	PDP-10	TENEX	133	205	85
MITAI	MIT	PDP-10	ITS	134	206	86

Fig. 1.1: Serving Hosts on the ARPA Network

4567	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NUL	DLE	DS		SP	&	-									0
0001	SOH	DC1	SOS				/		a	j	°	1	A	J		1
0010	STX	DC2	FS	SYN					b	k	s	2	B	K	S	2
0011	ETX	TM							c	l	t	3	C	L	T	3
0100	PF	RES	BYP	PN					d	m	u	4	D	M	U	4
0101	HT	NL	LF	RS					e	n	v	5	E	N	V	5
0110	LC	BS	ETB	UC					f	o	w	6	F	O	W	6
0111	DEL	IL	ESC	EOT					g	p	x	7	G	P	X	7
1000	GE	CAN							h	q	y	8	H	Q	Y	8
1001	RLF	EM							i	r	z	9	I	R	Z	9
1010	SMM	CC	SM		¢	!		:								
1011	VT	CU1	CU2	CU3	.	$	,	#	{	}	↳	↲				
1100	FF	IFS		DC4	<	*	%	@	≤	¤	↱	↴				
1101	CR	IGS	ENQ	NAK	(	)	_	'	(	)	[	]				
1110	SO	IPS	ACK		+	;	>	=	+	±	≥	≠				
1111	SI	IUS	BEL	SUB			¬	?	"	÷	•	•	–			

0	1	2	3	4	5	6	7

Code Structure

Fig. I.2: The EBCDIC Code

TN73-50-I(L.3)

4321	8 0	7 0	6 0	5 0	0	1	2	3	4	5	6	7
					0	0	0	0	1	1	1	1
					0	0	1	1	0	0	1	1
					0	1	0	1	0	1	0	1
0000					NUL	DLE	SP	0	@	P	`	p
0001					SOH	DC1	!	1	A	Q	a	q
0010					STX	DC2	"	2	B	R	b	r
0011					ETX	DC3	#	3	C	S	c	s
0100					EOT	DC4	$	4	D	T	d	t
0101					ENQ	NAK	%	5	E	U	e	u
0110					ACK	SYN	&	6	F	V	f	v
0111					BEL	ETB	'	7	G	W	g	w
1000					BS	CAN	(	8	H	X	h	x
1001					HT	EM	)	9	I	Y	i	y
1010					LF	SUB	*	:	J	Z	j	z
1011					VT	ESC	+	;	K	[	k	{
1100					FF	FS	,	<	L	\	l	|
1101					CR	GS	-	=	M	]	m	}
1110					SO	RS	.	>	N	^	n	~
1111					SI	US	/	?	O	_	o	DEL

Code Structure

Fig. L.3: The USASCII Code

ASCII DEC	ASCII OCT	ASCII HEX	SYMBOLS	EBCDIC HEX	EBCDIC DEC
0	0	(00)	NUI	(00)	00
1	1	(01)	SOH	(01)	01
2	2	(02)	STX	(02)	02
3	3	(03)	ETX	(03)	03
4	4	(04)	EOT	(37)	55
5	5	(05)	ENQ	(2D)	45
6	6	(06)	ACK	(2E)	46
7	7	(07)	BEL	(2F)	47
8	10	(08)	BS	(16)	22
9	11	(09)	HT	(05)	05
10	12	(0A)	LF	(25)	37
11	13	(0B)	VT	(0B)	11
12	14	(0C)	FF	(0C)	12
13	15	(0D)	CR	(0D)	13
14	16	(0E)	SO	(0E)	14
15	17	(0F)	SI	(0F)	15
16	20	(10)	DLE	(10)	16
17	21	(11)	DC1	(11)	17
18	22	(12)	DC2	(12)	18
19	23	(13)	DC3	(13)	19
20	24	(14)	DC4	(3C)	60
21	25	(15)	NAK	(3D)	61
22	26	(16)	SYN	(32)	50
23	27	(17)	ETB	(26)	38
24	30	(18)	CAN	(18)	24
25	31	(19)	EM	(19)	25
26	32	(1A)	SUB	(3F)	63
27	33	(1B)	CTI	(27)	39
28	34	(1C)	FS	(1C)	28
29	35	(1D)	GS	(1D)	29
30	36	(1E)	RS	(1E)	30
31	37	(1F)	US	(1F)	31

Fig. L.4: TELNET ASCII/EBCIDC Code Mapping

ASCII DEC	ASCII OCT	ASCII HEX	SYMBOLS	EBCDIC HEX	EBCDIC DEC
32	40	(20)	SP	(40)	64
33	41	(21)	!	(5A)	90
34	42	(22)	"	(7F)	127
35	43	(23)	#	(7B)	123
36	44	(24)	$	(5B)	91
37	45	(25)	%	(6C)	108
38	46	(26)	&	(50)	80
39	47	(27)	'	(7D)	124
40	50	(28)	(	(4D)	77
41	51	(29)	)	(5D)	93
42	52	(2A)	*	(5C)	92
43	53	(2B)	+	(4E)	78
44	54	(2C)	,	(6D)	109
45	55	(2D)	-	(60)	96
46	56	(2E)	.	(4B)	75
47	57	(2F)	/	(61)	97
48	60	(30)	0	(F0)	240
49	61	(31)	1	(F1)	241
50	62	(32)	2	(F2)	242
51	63	(33)	3	(F3)	243
52	64	(34)	4	(F4)	244
53	65	(35)	5	(F5)	245
54	66	(36)	6	(F6)	246
55	67	(37)	7	(F7)	247
56	70	(38)	8	(F8)	248
57	71	(39)	9	(F9)	249
58	72	(3A)	:	(7A)	122
59	73	(3B)	;	(5E)	94
60	74	(3C)	<	(4C)	76
61	75	(3D)	=	(7E)	126
62	76	(3E)	>	(6E)	110
63	77	(3F)	?	(6F)	111

Fig. L.4: TEINET ASCII/EBCIDC Code Mapping (Continued)

ASCII DEC	ASCII OCT	ASCII HEX	SYMBOLS	EBCDIC HEX	EBCDIC DEC	
64	100	(40)	@	(7C)	124	
65	101	(41)	A	(C1)	193	
66	102	(42)	B	(C2)	194	
67	103	(43)	C	(C3)	195	
68	104	(44)	D	(C4)	196	
69	105	(45)	E	(C5)	197	
70	106	(46)	F	(C6)	198	
71	107	(47)	G	(C7)	199	
72	110	(48)	H	(C8)	200	
73	111	(49)	I	(C9)	201	
74	112	(4A)	J	(D1)	209	
75	113	(4B)	K	(D2)	210	
76	114	(4C)	L	(D3)	211	
77	115	(4D)	M	(D4)	212	
78	116	(4E)	N	(D5)	213	
79	117	(4F)	O	(D6)	214	
80	120	(50)	P	(D7)	215	
81	121	(51)	Q	(D8)	216	
82	122	(52)	R	(D9)	217	
83	123	(53)	S	(E2)	226	
84	124	(54)	T	(E3)	227	
85	125	(55)	U	(E4)	228	
86	126	(56)	V	(E5)	229	
87	127	(57)	W	(E6)	230	
88	130	(58)	X	(E7)	231	
89	131	(59)	Y	(E8)	232	
90	132	(5A)	Z	(E9)	233	
91	133	(5B)	[	(AD)	173	
92	134	(5C)	\	(4A)	74	(BACK-SLASH)
93	135	(5D)	]	(BD)	189	
94	136	(5E)		(71)	113	(CARAT)
95	137	(5F)	_	(6D)	109	

Fig. L.4: TELNET ASCII/EBCIDC Code Mapping (Continued)

ASCII DEC	ASCII OCT	ASCII HEX	SYMBOLS	EBCDIC HEX	EBCDIC DEC	
96	140	(60)	`	(79)	121	(GRAVE)
97	141	(61)	a	(81)	129	
98	142	(62)	b	(82)	130	
99	143	(63)	c	(83)	131	
100	144	(64)	d	(84)	132	
101	145	(65)	e	(85)	133	
102	146	(66)	f	(86)	134	
103	147	(67)	g	(87)	135	
104	150	(68)	h	(88)	136	
105	151	(69)	i	(89)	137	
106	152	(6A)	j	(91)	145	
107	153	(6B)	k	(92)	146	
108	154	(6C)	l	(93)	147	
109	155	(6D)	m	(94)	148	
110	156	(6E)	n	(95)	149	
111	157	(6F)	o	(96)	150	
112	160	(70)	p	(97)	151	
113	161	(71)	q	(98)	152	
114	162	(72)	r	(99)	153	
115	163	(73)	s	(A2)	162	
116	164	(74)	t	(A3)	163	
117	165	(75)	u	(A4)	164	
118	166	(76)	v	(A5)	165	
119	167	(77)	w	(A6)	166	
120	170	(78)	x	(A7)	167	
121	171	(79)	y	(A8)	168	
122	172	(7A)	z	(A9)	169	
123	173	(7B)	{	(8B)	139	
124	174	(7C)	|	(4F)	79	(BAR/OR)
125	175	(7D)	}	(9B)	155	
126	176	(7E)	~	(5F)	95	(TILDE/NOT)
127	177	(7F)	DEL	(07)	7	

ASCII DEC	ASCII OCT	ASCII HEX	TELNET CONTROLS	EBCDIC HEX	EBCDIC DEC	
128	100	(80)	DATA-MARK	(80)	128	
129	101	(81)	BREAK	(38)	56	
130	102	(82)	NOP	(17)	23	IDLE
131	103	(83)	NOECHO	(14)	20	RESTORE
132	104	(84)	ECHO	(23)	35	
133	105	(85)	HIDE-YOUR-INPUT	(24)	36	BYPASS

Fig. I.4: TELNET ASCII/EBCDIC Code Mapping (Continued)

	EBCDIC		EBCDIC	ASCII
CINT	(4A)	= ESC	(27)	(1B)
CTL <	(4C)	= LEFT BRACKET	(AD)	(5B)
CTL >	(6E)	= RIGHT BRACKET	(BD)	(5D)
CTL (	(4D)	= LEFT BRACE	(8B)	(7B)
CTL)	(5D)	= RIGHT BRACE	(9B)	(7D)
CTL /	(61)	= BACK SLASH	(4A)	(C)
CTL "	(7F)	= CARAT	(71)	(5E)
CTL '	(7D)	= GRAVE	(79)	(60)
CTL 6	(F6)	= FS	(1C)	(1C)
CTL 7	(F7)	= GS	(1D)	(1D)
CTL 8	(F8)	= RS	(1E)	(1E)
CTL 9	(F9)	= US	(1F)	(1F)
CTL _	(6D)	= US	(1F)	(1F)
CTL ¬	(5F)	= DEL	(07)	(7F)
CTL @	(7C)	= NUL	(00)	(00)
CTL A	(C1)	= SOH	(01)	(01)
CTL B	(C2)	= STX	(02)	(02)
CTL C	(C3)	= ETX	(03)	(03)
CTL D	(C4)	= EOT	(37)	(04)
CTL E	(C5)	= ENQ	(2D)	(05)
CTL F	(C6)	= ACK	(2E)	(06)
CTL G	(C7)	= BEL	(2F)	(07)
CTL H	(C8)	= BS	(16)	(08)
CTL I	(C9)	= HT	(05)	(09)
CTL J	(D1)	= LF	(25)	(0A)
CTL K	(D2)	= VT	(0B)	(0B)
CTL L	(D3)	= FF	(0C)	(0C)
CTL M	(D4)	= CR	(0D)	(0D)
CTL N	(D5)	= SO	(0E)	(0E)
CTL O	(D6)	= SI	(0F)	(0F)
CTL P	(D7)	= DLE	(10)	(10)
CTL Q	(D8)	= DC1	(11)	(11)
CTL R	(D9)	= DC2	(12)	(12)
CTL S	(E2)	= DC3	(13)	(13)
CTL T	(E3)	= DC4	(3C)	(14)
CTL U	(E4)	= NAK	(3D)	(15)
CTL V	(E5)	= SYN	(32)	(16)
CTL W	(E6)	= ETB	(26)	(17)
CTL X	(E7)	= CAN	(18)	(18)
CTL Y	(E8)	= EM	(19)	(19)
CTL Z	(E9)	= SUB	(3F)	(1A)

Fig. L.5: 2741 Keyboard Control Character Mappings

	EBCDIC		EBCDIC	ASCII
CTL 1	(F1) =	BREAK	(38)	(81) - CIRCLE C
CTL 2	(F2) =	NOP	(17)	(82) - IDLE
CTL 3	(F3) =	NO ECHC	(14)	(83) - RESTORE
CTL 4	(F4) =	ECHO	(23)	(84)
CTL 5	(F5) =	HIDE YOU INPUT	(24)	(85) - BYPASS

DATA MARK (80) CANNOT BE ENTERED FROM THE KEYBOARD

THE FOLLOWING 2741 KEYBOARD CHARACTERS DO NOT HAVE A MEANING AS A CONTROL:

$ * * % ε
> - = ¬
. , : ;
! | ? ¢
SPACE
BACKSPACE
TAB

Fig. L.5: 2741 Keyboard Control Character Mappings (Continued)

```
•y  0 1 2 3 4 5 6 7 8 9 A B C D E F
x

4   . . . . . . . . . . . .¢...<.(.+.|.
5   .&. . . . . . . . . . .!.$.*.).;.¬.
6   .-./. . . . . . . . . . .,.%._.>.?.
7   . . . . . . . . . . . .:.#.@.'.=.".
8   . .a.b.c.d.e.f.g.h.i. . . . . . .
9   . .j.k.l.m.n.o.p.q.r. . . . . . .
A   . . .s.t.u.v.w.x.y.z. . . . . . .
B   . . . . . . . . . . . . . . . . .
C   . .A.B.C.D.E.F.G.H.I. . . . . . .
D   . .J.K.L.M.N.O.P.Q.R. . . . . . .
E   . . .S.T.M.V.W.X.Y.Z. . . . . . .
F   .0.1.2.3.4.5.6.7.8.9. . . . . . .
```

Hex Code X'xy' for Characters

```
••y  0 1 2 3 4 5 6 7 8 9
xx

06                 . . . . . . .
07   . . . . .¢...<.(.+.|.
08   .&. . . . . . . . . .
09   .!.$.*.).;.¬. ./. . .
10   . . . . . . . . .,.%._.
11   .>.?. . . . . . . . . .
12   . . .:.#.@.'.=.". .a.
13   .b.c.d.e.f.g.h.i. . .
14   . . . . . .j.k.l.m.n.
15   .o.p.q.r. . . . . . .
16   . . .s.t.u.v.w.x.y.z.
17   . . . . . . . . . . .
18   . . . . . . . . . . .
19   . . . .A.B.C.D.E.F.G.
20   .H.I. . . . . .J.
21   .K.L.M.N.O.P.Q.R. . .
22   . . . . . .S.T.U.V.
23   .W.X.Y.Z. . . . . .
24   .0.1.2.3.4.5.6.7.8.9.
25   . . . . . .
```

Decimal Code D'xxy' for Characters

```
HT   X'05' = D'005'   Horizontal Tab
LC   X'06' = D'006'   Lower Case
RES  X'14' = D'020'   Print Restore
NL   X'15' = D'021'   New Line
BS   X'16' = D'022'   Back Space
IL   X'17' = D'023'   Idle
BYP  X'24' = D'036'   Print Bypass
LF   X'25' = D'037'   Line Feed
UC   X'36' = D'054'   Upper Case
```

Hex Code X'xy' and Decimal Code D'xxy' Control Codes

Fig. I.6: 2741 Character and Control Codes

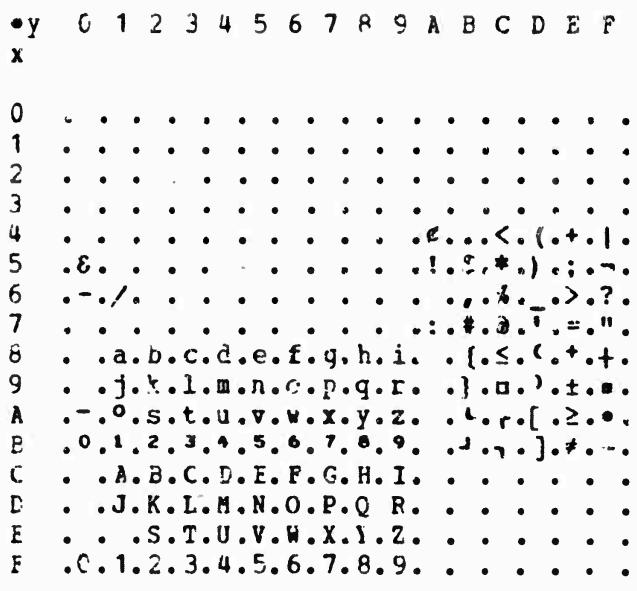
•y x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	.	.A	.B	.C	.D	.E	.F	.G	.H	.I	.	.	.	<	.	(	.	+	.	|
1	.	&	.J	.K	.L	.M	.N	.O	.P	.Q	.R	.	.	$	.	*	.	)	.	;	.	¬
2	.	-	.	/	.	S	.T	.U	.V	.W	.X	.Y	.Z	.	.	,	.	%	.	_	.	>	.	?
3	.	0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.	:	.	#	.	@	.	'	.	=	.	"
4	.	.	.A	.B	.C	.D	.E	.F	.G	.H	.I	.	.	.	<	.	(	.	+	.	|
5	.	&	.J	.K	.L	.M	.N	.O	.P	.Q	.R	.	.	$	.	*	.	)	.	;	.	¬
6	.	-	.	/	.	S	.T	.U	.V	.W	.X	.Y	.Z	.	.	,	.	%	.	_	.	>	.	?
7	.	0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.	:	.	#	.	@	.	'	.	=	.	"
8	.	.	.A	.B	.C	.D	.E	.F	.G	.H	.I	.	.	.	<	.	(	.	+	.	|
9	.	&	.J	.K	.L	.M	.N	.O	.P	.Q	.R	.	.	$	.	*	.	)	.	;	.	¬
A	.	-	.	/	.	S	.T	.U	.V	.W	.X	.Y	.Z	.	.	,	.	%	.	_	.	>	.	?
B	.	0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.	:	.	#	.	@	.	'	.	=	.	"
C	.	.	.A	.B	.C	.D	.E	.F	.G	.H	.I	.	.	.	<	.	(	.	+	.	|
D	.	&	.J	.K	.L	.M	.N	.O	.P	.Q	.R	.	.	$	.	*	.	)	.	;	.	¬
E	.	-	.	/	.	S	.T	.U	.V	.W	.X	.Y	.Z	.	.	,	.	%	.	_	.	>	.	?
F	.	0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.	:	.	#	.	@	.	'	.	=	.	"

Hex Code X'xy'

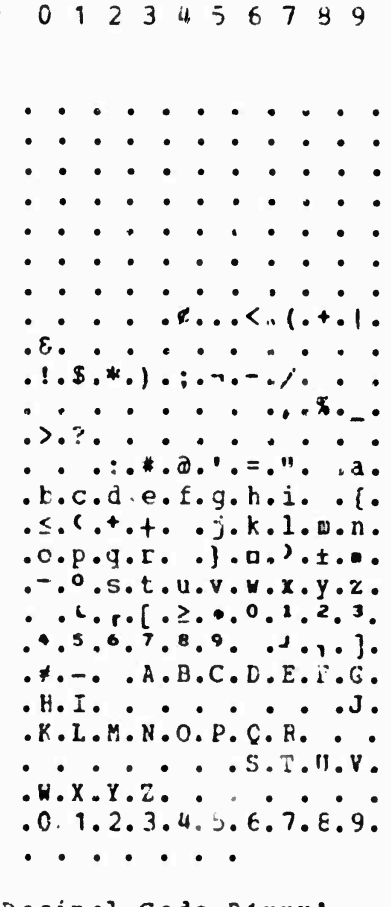
••y xx	0	1	2	3	4	5	6	7	8	9
00	.	.A	.B	.C	.D	.E	.F	.G	.H	.I
01	.	.	.	<	.	(	.	+	.	|	.	&	.J	.K	.L
02	.M	.N	.O	.P	.Q	.R	.	.	$	.	*	.	)	.
03	.	;	.	¬	.	-	.	/	.	S	.T	.U	.V	.W	.X
04	.Y	.Z	.	.	,	.	%	.	_	.	>	.	?	.	0	.1
05	.2	.3	.4	.5	.6	.7	.8	.9	.	:	.	#	.
06	.	@	.	'	.	=	.	"	.	.	A	.B	.C	.D	.E
07	.F	.G	.H	.I	.	.	.	<	.	(	.	+	.	|
08	.	&	.J	.K	.L	.M	.N	.O	.P	.Q	.R
09	.	.	$	.	*	.	)	.	;	.	¬	.	-	.	/	.	S	.T
10	.U	.V	.W	.X	.Y	.Z	.	.	,	.	%	.	_	.
11	.	>	.	?	.	0	.1	.2	.3	.4	.5	.6	.7
12	.8	.9	.	:	.	#	.	@	.	'	.	=	.	"	.	.	A	.
13	.B	.C	.D	.E	.F	.G	.H	.I	.	.	.
14	.	<	.	(	.	+	.	|	.	&	.J	.K	.L	.M	.N
15	.C	.P	.Q	.P	.	$	.	*	.	)	.	;	.	¬
16	.	-	.	/	.	S	.T	.U	.V	.W	.X	.Y	.Z
17	.	.	,	.	%	.	_	.	>	.	?	.	0	.1	.2	.3
18	.4	.5	.6	.7	.8	.9	.	:	.	#	.	@	.	'
19	.	=	.	"	.	.	A	.B	.C	.D	.E	.F	.G
20	.H	.I	.	.	.	<	.	(	.	+	.	|	.	&	.J
21	.K	.L	.M	.N	.O	.P	.Q	.R	.	$	.
22	.	*	.	)	.	;	.	¬	.	-	.	/	.	S	.T	.U	.V
23	.W	.X	.Y	.Z	.	.	,	.	%	.	_	.	>	.	?
24	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9
25	.	:	.	#	.	@	.	'	.	=	.	"

Decimal Code D'xxy'

Fig. L.7: Code for Characters on a PN Print Train



Hex Code X'xy'



Decimal Code D'xxy'

Fig. L.8: Code for Characters on a PN Print Train

References

1. BOLT, BERANEK & NEWMAN Inc. Report Number 1763, January 1969, AD-682 905.

2. BOLT, BERANEK & NEWMAN Inc. Interface Message Processor, Specifications for the Interconnection of a HOST and an IMP. Report Number 1822, February 1971 Revision.

3. BRYAN, P.Z. IMP Interface Specification. Private Communication, December 1970.

4. CARR, C., CROCKER, S. & CERF, V. HOST-HOST Communication Protocol in the ARPA Network. AFIPS Conference Proceedings, 1970 Spring Joint Computer Conference, Atlantic City N.J., 589-597.

5. COLE, G.D. Performance Measurements on the ARPA Network. IEEE Transactions on Communications, Vol COM-20, No 3, June 1972, 630-636.

6. CROCKER, S. D. HOST/HOST Protocol for the ARPA Network. Document Number 1, 1971 Revision.

7. CROCKER, S., HEAFNER, J., METCALFE, R. & POSTEL, J. Function Oriented Protocols for the ARPA Network. AFIPS Conference Proceedings, 1972 Spring Joint Computer Conference, 271-280.

8. HEART, F., KAHN, R., ORNSTEIN, S., CROWTHER, W. & WALDEN, I. The Interface Message Processor for the ARPA Computer Network. AFIPS Conference Proceedings, 1970 Spring Joint Computer Conference, Atlantic City N.J., 551-567.

9. IBM CP-67/CMS User's Guide. Cambridge Scientific Centre, IBM Data Processing Division, 320-2015, July 1969 Revision.

10. KIRSTEIN, P.T. On the Status of the ARPA Network, Summer 1972. INDRA Note 275, University of London Institute of Computer Science, November 1972.

11. McKENZIE, A. Proposed Change in IMP-to-HOST Protocol. Network Working Group, Request for Comment 312, NIC 9342, 22 March 1972.

12. McQUILLAN, J., CROWTHER, W., COSELL, B., WALDEN, D. & HEART, F. Improvements in the Design and Performance of the ARPA Network. AFIPS Conference Proceedings, 1972 Fall Joint Computer Conference, 741-754.

13. ORNSTEIN, S., HEART, F., CROWTHER, W., RISING, H., RUSSEL, S. & MICHEL, A. The Terminal IMP for the ARPA Computer Network. AFIPS Conference Proceedings, 1972 Spring Joint Computer Conference, 243-254.

14. O'SULLIVAN, T.C. TELNET Protocol. Network Working Group, Request for Comment 158, NIC 6768, 19 May 1971.

15. POSTEL, J. A Proferred Official Connection Protocol. Network Working Group, Request for Comment 165, NIC 6779, 25 May 1971.

16. ROBERTS, L.G. A Forward Look. Journal of the Armed Forces Communications and Electronics Association, Vol XXV, No 12, August 1971, 77-81.

17. ROBERTS, L.G. & WESSLER, B.D. Computer Network Development to Achieve Resource Sharing. AFIPS Conference Proceedings, 1970 Spring Joint Computer Conference, 543-549.

18. SPOONER, C.P. A Software Architecture for the 70's: Part 1 - The General Approach. Software Practice and Experience, Vol 1, 1971, 5-37.

19. WALDEN, D.C. A System for Interprocess Communication in a Resource Sharing Network. Communications of the ACM, Vol 15, No 4, April 1972, 221-230.

20. WHITE, J.E. An NCP for the ARPA Network. Computer Research Laboratory, UCSB, California, 21 December 70.